SOFTWARE

AMIGA

HiSoft

Devpac Assembler

deutsch

Entwicklungspaket mit integriertem Editor/Assembler, symbolischem Debugger und schnellem Linker zum Einbinden von Hochsprachen-Modulen. Erzeugt direkt ausführbare Programme.

3½"-Diskette



HiSoft



3½"-Diskette

AMIGA

Entwicklungspaket mit integriertem Editor/Assembler, symbolischem Debugger und schnellem Linker zum Einbinden von Hochsprachen-Modulen. Erzeugt direkt ausführbare Programme.



3½"-Diskette

HiSoft

Markt&Technik Verlag Aktiengesellschaft · Hans-Pinsel-Straße 2 · D-8013 Haar bei München

15 14 13 12 11 10 9 8 7 6 5 4 3 2
91 90 89 88

Bestell-Nr. 51656

Einbandgestaltung: Grafikdesign Heinz Rauner
Druck: Bosch, Landshut
Printed in Germany

Inhaltsverzeichnis

Vorwort

Kapitel 1
Einführung

Kapitel 2
Der kombinierte Schirm-Editor und Makro-Assembler

Eingabe von Text und Cursor-Bewegungen

Löschen von Text

Vorwort des Übersetzers

Bevor ich das Amiga-Assembler-Buch schrieb, habe ich acht verschiedene Assembler getestet und davon drei, nämlich Metacomco, SEKA und *Devpac* in die engere Wahl gezogen. Nach einem ausführlichen Test dieser drei stand für mich als eindeutiger Sieger *Devpac* von HiSoft fest. Ich habe daraufhin die zahlreichen Listings des Buches mit *Devpac* entwickelt und immer gehofft, daß auch die Leser meiner Meinung sein werden. Deshalb freut es mich besonders, daß sich ein so führender Software-Spezialist wie Markt&Technik dafür entschieden hat, dieses Produkt in sein Angebot aufzunehmen, womit meine Einschätzung dieses Assemblers bestätigt wurde.

Das war für mich auch der Grund, die Übersetzung des Handbuches zu übernehmen, obwohl ich sonst viel lieber kreativ als Programmierer und Autor arbeite.

Ganz konnte ich diese »Kreativität« auch bei der Übersetzung nicht unterdrücken, weshalb dies keine Wort-für-Wort-Übersetzung geworden ist. Ich habe einige Inkonsistenzen beseitigt und einige Passagen, die meiner Meinung nach Einsteiger nicht auf Anhieb verstehen, etwas klarer ausgedrückt, als es dies das Original tut. Andererseits bin ich schon der Ansicht, daß im Original einige Passagen sehr ausführlich beschrieben sind, hier habe ich dennoch nichts gekürzt und nichts weggelassen.

Einige Worte habe ich nicht übersetzt. Dabei handelt es sich um Fachausdrücke aus der Assembler-Programmierung und um Begriffe aus der Amiga-Welt. Die Assembler-Sprache selbst ist nun einmal englischer Klartext, da würde ein Eindeutschen nur die Verbindung zur Praxis aufheben. Sinngemäßes gilt für die Amiga-Welt. Die vielen hundert Funktionen, die Ihnen das Betriebssystem zur Verfügung stellt, haben englische Namen. Das gilt auch für alle CLI-Kommandos. Hier würde zum Beispiel das Wort *Inhaltsverzeichnis* anstatt *Directory* die Verbindung zu Kommandos, wie *dir* oder *makedir* aufheben.

Eines fehlt im Buch ganz, nämlich die Erläuterung sehr häufig gebrauchter Namen. *Devpac* ist das Kürzel für »Development Package« also Entwicklungspaket. Tatsächlich erhalten Sie mit *Devpac* ein Paket, bestehend aus Editor, Assembler und Debugger (und einigen Zugaben).

MonAm oder *MonAmiga* ist das Kürzel für Monitor Amiga. Hierbei handelt es sich um einen symbolischen Debugger. Ich nenne *MonAm* auch Mon Ami (mein Freund), weil er mir schon aus so mancher Klemme geholfen hat.

Bliebe noch *GenAm* oder *GenAmiga*. Das Gen heißt Generator, genau Programm-Generator, womit die Kombination von Editor und Makro-Assembler gemeint ist. Da diese beiden wichtigsten Werkzeuge als integrierte Einheit ständig im RAM stehen,

reicht ein Tastendruck im Editor, um den Assembler aufzurufen. Dieser merkt sich bis zu 30 Fehlerstellen, womit man dann im Editor von Fehlerzeile zu Fehlerzeile gehen und verbessern kann. Was dann jeweils falsch ist, steht sogar in der Statuszeile.
Dieser Bedienkomfort, verbunden mit hohem Tempo und allen Leistungsmerkmalen guter Assembler, hat mich für *Devpac* begeistert und tut es immer noch.
Ich wünsche Ihnen viel Erfolg bei der Entwicklung Ihrer Programme. Die besten Voraussetzungen dafür haben Sie mit Ihrer Entscheidung für *Devpac* bereits geschaffen.
Happy Programming!

Ihr Peter Wollschlaeger

Kapitel

1

Einführung in das Programm

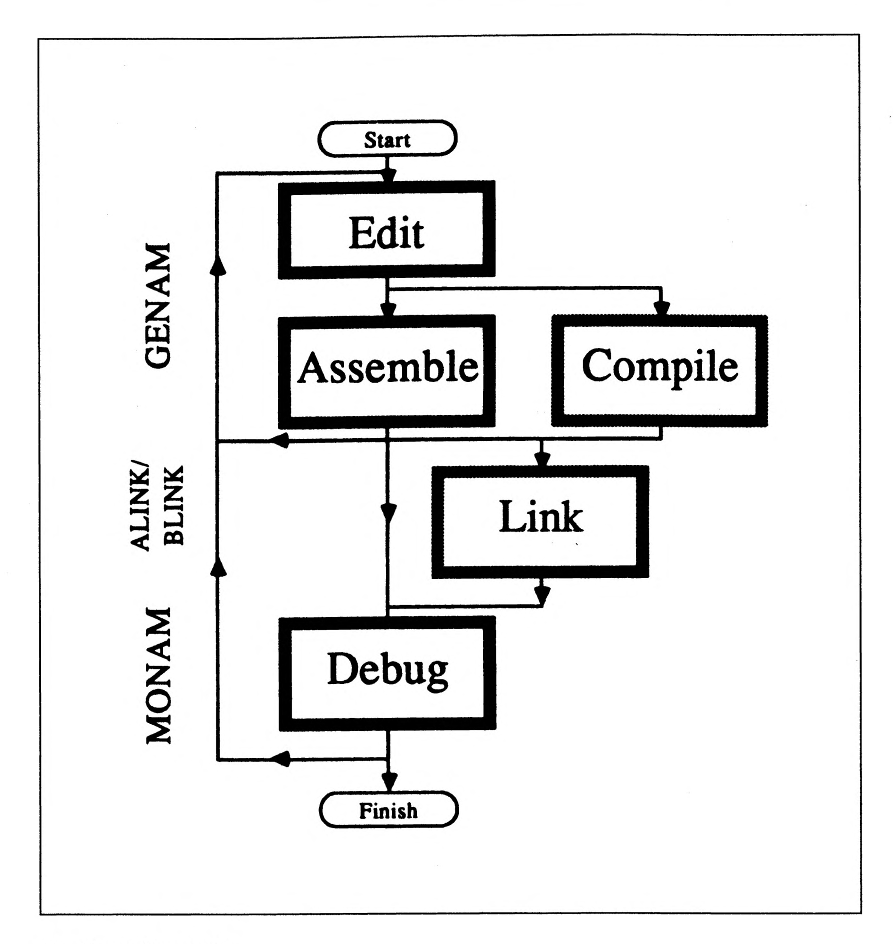
1 Einführung in das Programm

1.1 Copyright

Die Software von *Devpac* wird auf einer nicht kopiergeschützten Diskette ausgeliefert. Software und Handbuch unterliegen dem deutschen Urheberrecht. Jegliche Vervielfältigung, auch auszugsweise, ist nur nach schriftlicher Genehmigung des Verlages gestattet. Hiervon ausgenommen ist lediglich das Anfertigen von Sicherungsdisketten ausschließlich für den eigenen Bedarf. Die Weitergabe an Dritte ist nicht zulässig.

1.2 Der Entwicklungs-Zyklus

Das *Devpac* Entwicklungspaket dient zur Eingabe des Assembler-Quellentextes, zur Assemblierung in Maschinen-Code und zum Debuggen, wenn ein Programm nicht (oder nicht so wie geplant) laufen sollte. Abhängig von der jeweiligen Anwendung können Sie auch einen Linker benutzen, um zum Beispiel mehrere Module zusammenzubinden. Auch Module, die von Compilern höherer Programmiersprachen generiert wurden, können so mit Assembler-Modulen gebunden werden.
Der normale Entwicklungs-Zyklus kann am besten mit einem Diagramm wie dem folgenden dargestellt werden. *Devpac* wurde entwickelt, um diesen Ablauf so schnell und effektiv wie möglich bewältigen zu können.
Die Linkfunktion ist optional, der Assembler kann auch sofort ausführbaren Code generieren. Natürlich wird ein Compiler (wie im Bild angedeutet) nicht mitgeliefert, es können aber alle gängigen verwendet werden. Als Linker muß der Standard-Amiga-Linker ALINK eingesetzt werden oder das Public-Domain-Programm Blink (sprich B-Link). Blink finden Sie auf der Diskette, die Beschreibung dazu im Textfile BLINK.DOC.



Der Entwicklungszyklus

1.3 Inhalt der Diskette

Die beigefügte 3½″-Diskette ist eine Standard-Workbench-Diskette, die so konfiguriert ist, daß der Amiga nach dem Einschalten (oder einem Reset) im CLI ist. Zusätzlich zu den üblichen System-Files finden Sie (mindestens) diese:

Programme (im Unter-Directory C/):

GenAm	der Bildschirmeditor und Makro-Assembler
MonAm	der Disassembler/Debugger
GenInst	das Installationsprogramm für GenAm

BLink	der Public-Domain-Linker
Checkl.2	stellt sicher, daß Kickstart 1.2 eingesetzt wird

Textfiles:

readme	Die neuesten Informationen über Devpac
blink.doc	Die Bedienungsanleitung zu BLink

Sonstige Dateien

include	enthält alle notwendigen Konstanten und Makros, Einzelheiten dazu siehe Anhang C
examples	Die Quelltexte und Objekt-Files zu verschiedenen Beispiel-Programmen

Anmerkung

Dieses Handbuch macht keinen Versuch, die 68000-Programmierung zu lehren oder auf Einzelheiten des 68000-Befehlssatzes einzugehen.
Auf geeignete Literatur wird im Anhang verwiesen.

1.4 Einführung in das CLI

Wie die meisten Entwicklungstools ist auch *Devpac* für den Betrieb im CLI (Command Line Interface) ausgelegt. Wenn Sie die Diskette von der Workbench aus inspizieren, so werden Sie deshalb nicht sehr viele Icons sehen (genauer gesagt, keine von *Devpac*-Files).
Das CLI ist für den Normalanwender versteckt, aber als Programmierer brauchen Sie es. Die mitgelieferte Diskette ist eine modifizierte Workbench-Diskette. Wenn Sie nach dem Einschalten (beim Amiga 1000 nach dem Durchlauf der Kickstart-Diskette) nach der Workbench-Diskette gefragt werden, so legen Sie die *Devpac*-Diskette ein.
Wenn Ihr Amiga schon mit einer anderen Workbench-Diskette gestartet wurde, so warten Sie eventuelle Disk-Aktivitäten ab, legen dann die *Devpac*-Diskette ein, und machen Sie Reset, was durch gleichzeitiges Drücken der Ctrl-Taste und der beiden Amiga-Tasten erreicht wird.

Das CLI entspricht der Benutzeroberfläche anderer nicht grafisch orientierter Computer, wie zum Beispiel MS-DOS oder CP/M. Es verlangt die Eingabe von Kommandos wie

```
dir  [Return]
```

um eine Directory-Anzeige zu erhalten.

1.5 Die Readme-Datei

Wie alle HiSoft-Produkte wird auch *Devpac* laufend verbessert, und die neuesten Informationen, die noch nicht in das Handbuch aufgenommen werden konnten, werden dann in der Readme-Datei bekanntgegeben. Wenn Sie noch nicht im CLI sind, so gehen Sie vor wie im Punkt 1.4 geschildert und tippen dann

```
type readme [Return]
```

Das (durchlaufende) Display kann durch Drücken der rechten Maustaste angehalten werden. Nachdem die Taste losgelassen wurde, läuft der Text weiter ab.

1.6 Immer eine Sicherungskopie anlegen!

Bevor Sie *Devpac* einsetzen, sollten Sie von der mitgelieferten Diskette eine Kopie erstellen und das Original an einem sicheren Platz verwahren. Es ist nicht kopiergeschützt, um Ihnen ein einfaches Arbeiten zu ermöglichen. Bevor Sie mit dem Kopieren beginnen, aktivieren Sie den Schreibschutz Ihrer Originaldiskette, um ihre versehentliche Zerstörung zu vermeiden. Die genaue Kopiermethode hängt davon ab, wieviele Laufwerke Sie besitzen, aber alle Methoden machen es erforderlich, daß Sie im CLI sind und mit dem schreibgeschützten Original im internen Laufwerk eingelegt beginnen. Wenn Sie noch nicht im CLI sind, verfahren Sie zuerst nach Punkt 1.4.

Amiga mit einem Disketten-Laufwerk

Geben Sie ein

```
diskcopy df0: to df0:  [Return]
```

Dann folgen Sie den Anweisungen, die auf dem Bildschirm erscheinen. Die »disk to copy from« ist unsere Master-Diskette, die »disk to copy to« sollte eine neue, leere Diskette sein.

Amiga mit zwei Disketten-Laufwerken

Geben Sie ein

```
diskcopy df0: to df1:  [Return]
```

Legen Sie dazu die neue, leere Diskette in das externe Laufwerk. Alles weitere erfahren Sie auf dem Bildschirm.
Nun haben Sie eine Sicherungskopie. Warten Sie ab, bis alle Diskettenaktivitäten beendet sind, werfen Sie die Diskette(n) aus und verwahren Sie das Original an einem sicheren Ort. Legen Sie nun die neue Kopie ein und erzeugen Sie einen Reset. (Ctrl und beide Amiga-Tasten gleichzeitig.)
Im nächsten Abschnitt folgt eine kurze Einführung über den Umgang mit *GenAm* und *MonAm* und darauf dann Hinweise zum Erstellen einer Arbeitsdiskette.

1.7 Eine Kurzeinführung

Das folgende Kapitel ist eine kurze Einführung mit dem Ziel Ihnen zu zeigen, wie einfach und schnell das Editieren, Assemblieren und Debuggen mit *Devpac* ist.
In diesem Tutorial werden wir ein einfaches Programm assemblieren und debuggen. Das Programm selbst soll nur einen Text auf dem Bildschirm ausgeben.
Um diesem Tutorial folgen zu können, müssen Sie bereits im CLI sein. Wenn dies nicht der Fall sein sollte,so warten Sie alle Diskettenaktivitäten ab und werfen alle Disketten aus. Drücken Sie die Tasten Ctrl und beide Amiga-Tasten gleichzeitig und legen dann Ihre laut Abschnitt 1.6 erstellte Kopie ein.

Um die Arbeit zu erleichtern, wird das Directory, in dem sich der Quelltext befindet, zum aktuellen Directory gemacht. Tippen Sie dazu

```
cd examples [Return]
```

Als nächstes laden Sie den Editor/Assembler durch die Eingabe von

```
genam  [Return]
```

Nach kurzer Zeit erscheint ein leeres Fenster. Um nun ein File zu laden, drücken Sie die rechte Maustaste, bewegen dann den Mauszeiger auf das Project-Menü und wählen dort »Load« aus. Es erscheint ein File-Requester. Klicken Sie das Namenfeld an und geben Sie den zu ladenden File-Namen ein, hier

```
demo.s [Return].
```

Wenn das File geladen ist, wird das Fenster die ersten Programmzeilen zeigen. Wenn Sie einen schnellen Blick auf das ganze Programm werfen wollen, können Sie die Tastenkombination Shift und Pfeil-unten oder Ctrl-C drücken, um eine Seite weiterzublättern.
Bei den meisten kurzen Programmen ist es am besten, wenn man sie versuchsweise assembliert, ohne gleich List-Files und Object-Files zu erzeugen. Damit ist eine schnelle Kontrolle auf Syntax-Fehler, Tippfehler und ähnliches möglich. Obwohl »demo.s« keine Fehler enthält, tun wir das auch.
Bewegen Sie den Mauszeiger auf das Options-Menü und wählen Sie »Assemble« aus. Es erscheint ein Requester, der im Feld »binary file« nichts anzeigt, was gewollt ist. Im Feld Listing steht auch nichts, auch das ist erwünscht. Klicken Sie OK an, und das Assemblieren beginnt.
Während des Assemblierens wird einige Male auf die Diskette zugegriffen, denn es werden mehrere Include-Files geladen, in denen Konstanten und Makros definiert sind.
Nach kurzer Zeit ist das Assemblieren beendet und die Meldung »`no errors`« (keine Fehler) erscheint. Drücken Sie nun irgendeine Taste oder eine Maustaste, um wieder in den Editor zu gelangen. Jetzt, wo wir wissen, daß keine Syntax- oder Semantik-Fehler aufgetreten sind, können wir richtig assemblieren. Wählen Sie also wieder Assemble aus dem Options-Menü.
Wenn der Requester wieder erscheint, klicken Sie das Feld »binary name« an und geben

```
ram:demo
```

ein, um das Binär-File anzugeben. Die Umleitung auf die RAM-Disk erfolgt nur aus Geschwindigkeitsgründen. Ein Klick auf OK startet das Assemblieren. Diesmal wird es etwas länger dauern, weil nun auch noch das Binär-File erstellt werden muß. Drücken Sie nach Beendigung wieder eine Taste, um in den Editor zu gelangen, und wählen Sie von dort »Quit« aus dem Project-Menü. Falls Sie zwischenzeitlich im Quelltext etwas geändert haben sollten, wird ein Alarm-Requester erscheinen. Klicken Sie dort auf OK (Änderung wird nicht gespeichert) und Sie sind auch dann wieder im CLI.
Um das Programm zu testen, tippen Sie

```
ram:demo  [Return]
```

Das Programm sollte nun arbeiten wie erwartet und einen Text ausgeben.

Obwohl das Programm funktioniert, sollten wir nun *MonAm*, den Debugger, einsetzen, um uns Schritt für Schritt durchzuarbeiten. Dazu tippen Sie ein

```
monam    [Return]
```

Wenn der Debugger geladen ist, wird er nach einem Filenamen fragen. Geben Sie ein

```
ram:demo    [Return]
```

Er antwortet »`Symbols loaded`« (Symboltabelle geladen) und fragt dann nach einer Kommandozeile. Da diese hier nicht gebraucht wird, geben Sie nur [Return] ein. *MonAm* setzt einen Breakpoint (Stop-Marke) auf den ersten Befehl des Programms und beginnt dann. Die Nachricht »`Exception: Breakpoint`« wird erscheinen, dazu noch das Haupt-Display.
Im obersten Fenster dieses Displays sind die Register-Inhalte angezeigt, hier hauptsächlich Nullen. Im zweitem Fenster steht das Speicher-Display. Das dritte Fenster bringt ein Disassembler-Listing, im untersten Fenster steht »Command:«.
Das Disassembler- und das Speicher-Fenster zeigen den Bereich ab dem Programmzähler (PC). Zu diesem Zeitpunkt in unserem Fall sollten Sie den Befehl

```
MOVEA.L    #dosname,A1
```

sehen. Weil die Labels im Binär-File abgelegt wurden, erscheinen sie automatisch auch hier. Schauen wir uns den Bereich um »dosname« herum an. Tippen Sie `M` wie Modifizieren und dann

```
.dosname   [Return]
```

Der Punkt ist wichtig, und der Name muß in Kleinschrift eingegeben werden. Die Fenster werden aktualisiert, und das Disassembler-Fenster wird andere Befehle anzeigen, aber im Speicher-Fenster sollten Sie die ASCII-Codes und den Text »dos.library« sehen können. Diesen Speicherbereich haben wir uns nun angesehen, weshalb wir wieder auf den PC-Stand gehen wollen. Drücken Sie die Tabulator-Taste, um das Display wieder auf den vorherigen Stand zu bringen.
Um den MOVEA-Befehl auszuführen, tippen Sie Ctrl-Z. Der Schirm wird aktualisiert, um die neuen Werte von PC, A1 und A7 anzuzeigen. Wenn Sie wieder Ctrl-Z drücken, wird der MOVEQ-Befehl ausgeführt.Der Wert von D0 sollte eine Long-Null sein.Tippen Sie wieder Ctrl-Z, und das Register A6 wird mit einem Zeiger geladen.
Jetzt sollten Sie ein JSR, einen Systemaufruf zum Öffnen der DOS-Library, sehen. Es würde viel Zeit kosten, diese Routine in Einzelschritten zu durchlaufen. Deshalb sollten Sie jetzt Ctrl-T eingeben, um dieses Unterprogramm auszuführen. Es sollte einen Wert ungleich Null im Register D0 zurückgeben. Das wird von den nächsten beiden Befehlen getestet, die Sie wieder mit Ctrl-Z abarbeiten können.

Die nächsten Anweisungen speichern Register D0 in der Variablen _DOSBase und laden dann Register A6 mit demselben Wert. Tippen Sie also zweimal Ctrl-Z. Danach folgt wieder ein Unterprogrammaufruf (welcher eine passende Output-Handle holt), und auch dieser sollte in einem Stück mittels Ctrl-T ausgeführt werden und einen Wert in D0 zurückgeben. Nun sind wir soweit, daß wir einen Text ausgeben können, weshalb es an dieser Stelle nützlich ist, sich die Workbench-Screen anzusehen. Dazu drücken Sie gleichzeitig die linke Amiga-Taste und N. Um zur *MonAm*-Screen zurückzukehren, nehmen Sie die linke Amiga-Taste und M.
Wenn Sie schon die Workbench-Screen angeklickt haben, dann haben Sie damit das *MonAm*-Fenster deaktiviert (sein Titel erscheint in Grau). Um wieder *MonAm*-Befehle eingeben zu können, müssen Sie das *MonAm*-Fenster an einer beliebigen Stelle anklicken.
Die Ausgabe wird mit 5 Befehlen vorbereitet, deshalb drücken Sie jetzt Ctrl-Z fünfmal, bis Sie bei einem JSR sind. Dieser Aufruf gibt Text aus. Tippen Sie nun Ctrl-T. Wenn Sie nun zur Workbench-Screen wechseln, wie eben beschrieben, sollten Sie den Text sehen können.
Der interessante Teil unseres Programms ist damit beendet, so ist es wohl das beste, es jetzt mittels Ctrl-R bis zum Ende laufen zu lassen. *MonAm* wird antworten »`Exception: Process terminated`«. Sie drücken dann Ctrl-C, um *MonAm* zuverlassen.

1.8 Einrichten einer Arbeitsdiskette

Obwohl wir eine Sicherungskopie haben (die haben Sie doch erstellt, oder?), hat diese nicht genügend Kapazität, um effektiv programmieren zu können. Deshalb sollte eine Arbeitsdiskette angelegt werden. Die genaue Vorgehensweise hängt von Ihrer Hardware-Konfiguration ab.
Auf der mitgelieferten Diskette wurde schon einiges gelöscht (speziell Fonts und Sprachausgabe), um Platz zu schaffen.

System mit einem Laufwerk

Bei Systemen mit nur einem Laufwerk ist es das beste, möglichst viel von der Workbench-Disk zu löschen. Dafür kommen zunächst alle .info-Files (welche für die Icon-Darstellung gebraucht werden), die Clock (Uhr) und die Directories Trashcan, Empty und Fonts in Frage. Die folgenden Kommandos sind (auf einer weiteren Sicherungskopie) erforderlich:

```
cd df0: [Return]
delete  #?.info [Return]
delete  trashcan all [Return]
delete  empty all [Return]
```

Danach sind ungefähr 20% der Disk für Ihre Quelltexte frei. Wenn es dann immer noch eng ist, können Sie das Examples-Directory löschen, selten genutzte Include-Files und noch einige der Kommandos im C-Directory. (Sie könnten eine Fehlermeldung sehen, wenn Sie das Font-Directory löschen, dies ist aber nicht von Bedeutung.)

System mit zwei Laufwerken

Bei Systemen mit zwei Laufwerken wird empfohlen, die Workbench-Disk von allen Quell-Files freizuhalten und diese Files zusammen mit den Include-Files auf der Diskette im externen Laufwerk abzulegen. Dazu sind die folgenden Kommandos (auf einer weiteren Sicherungskopie) erforderlich:

Wichtig: Der erste Befehl formatiert eine neue Diskette im externen Laufwerk und vernichtet damit alle darauf befindlichen Daten. Wenn Sie eine vorformatierte Diskette benutzen, brauchen Sie diesen Befehl nicht auszuführen.

```
format drive df1: name source [Return]
```

Legen Sie eine neue Diskette in das externe Laufwerk und geben dann [Return] ein, um sie zu formatieren. Nun folgt:

```
cd df0:  [Return]
makedir df1:include  [Return]
makedir df1:examples  [Return]
copy include to df1:include all  [Return]
copy examples to df1:examples all [Return]
delete include all  [Return]
delete examples all  [Return]
cd df1:  [Return]
```

Die Kommandos schaffen Directories auf dem externen Drive für die Beispiel-Programme (in examples) und die Include-Files. Dann löschen Sie diese von Ihrer Arbeitskopie im internen Laufwerk. Der letzte Befehl schaltet auf das externe Laufwerk um, so daß Sie Ihre Quellfiles einfacher (ohne Angabe des Pfadnamens) editieren können. Es ist oft nützlich, so ein Kommando in das Startup-File (siehe Anhang D) einzubauen.

Für Systeme mit Festplatte

Festplattenbesitzer sollten zuerst von der Harddisk booten und nicht von der *Devpac*-Diskette. Danach sollte die *Devpac*-Diskette in das interne Laufwerk eingelegt werden und die folgenden Befehle ausgeführt werden, um die Programme auf die Harddisk zu kopieren:

```
copy df0:c/genam to c:  [Return]
copy df0:c/monam to c:  [Return]
copy df0:c/geninst to c:  [Return]
copy df0:c/blink to c: [Return]
```

Als nächstes sollten die beiden Sub-Directories kopiert werden, doch dieses hängt von Ihren Anforderungen ab. Sie könnten sich zum Beispiel ein extra *Devpac*-Directory anlegen.
Was immer Sie wählen, Sie müssen die Include- und Example-Directories komplett kopieren. Wenn Sie sich für ein extra *Devpac*-Directory entschieden haben, sind folgende Kommandos erforderlich (angenommen Ihre Harddisk heißt dh0:)

```
makedir dh0:devpac [Return]
copy df0:include to dh0:devpac all [Return]
copy df0:examples to dh0:devpac all [Return]
```

Beachten Sie, daß Sie die INCDIR-Directive entsprechend ändern müssen, wenn Sie das Include-Directory in ein anderes Directory gepackt haben.

Wenn Sie das CLI bisher noch nicht benutzt haben, ist es empfehlenswert, erst den Anhang D zu lesen, bevor Sie fortfahren.

Kapitel

2

Der Editor/ Assembler

2 Der Editor/Assembler

2.1 Einführung

Für die Eingabe und das Assemblieren eines Programms brauchen Sie einen Editor und einen Assembler. *GenAmiga* kombiniert beide Funktionen in einem integrierten Programm mit einem echten Bildschirm-Editor und einem schnellen Assembler, der die Motorola-Spezifikation voll erfüllt. Durch die Kombination dieser beiden Programme kann die Fehlerkorrektur und das Ändern so schnell wie möglich erfolgen, ohne daß man auf die (vergleichsweise) langsame Diskette zugreifen muß.

Um *GenAmiga* zu starten, tippen Sie einfach

```
genam [Return]
```

vom CLI aus. Nach dem Laden ist eine Menü-Zeile verfügbar und ein leeres Fenster ist für die Eingabe und Assemblierung Ihres Programms geöffnet. Sie können wahlweise noch eine Kommandozeile mit angeben, die den File-Namen eines zu editierenden Programms und verschiedene Optionen enthalten kann.

Option	Beispiel
`-s` Setze Arbeitsbereich für Editor	`-s100` (ergibt 100 Kbyte)
`-x` Setze Maximum XREFs	`-x250` (250 XREF erlaubt)
`+c` Groß-Klein-Buchstaben-Unterscheidung	
`-c` Großbuchstaben=Kleinbuchstaben	
`+l` Erzeugt linkbaren Code	
`-l` Erzeugt ausführbaren Code	

Die Bedeutung der Optionen wird Ihnen klarer, wenn Sie den Rest dieses Kapitels gelesen haben. Zum Beispiel das Kommando `genam examples/demo.s -s50 +c -l` wird *GenAmiga* starten, den File demo.s in einen Arbeitspuffer von 50 Kbyte laden, beim Assemblieren Groß-Klein-Schrift unterscheiden und ausführbaren Code erzeugen.

Die Beschreibung des Editor/Assemblers erfolgt in zwei getrennten Abschnitten. Beginnen wir mit dem Editor.

2.2 Der Bildschirm-Editor

2.2.1 Einführung

Ein Text-Editor ist ein Programm, das Ihnen erlaubt, Textzeilen in den Speicher einzugeben, zu ändern, sie auf der Diskette zu speichern und wieder von dort zu laden. Es gibt zwei Arten von Editoren, zum einen Zeilen-Editoren, die jede Zeile einzeln behandeln und sehr schwierig zu bedienen sind, und zum anderen Schirm-Editoren, die Ihren Text auf einmal auf dem Schirm abbilden. Diese sind wesentlich einfacher zu bedienen.
Der Editor-Teil von *GenAmiga* ist ein Bildschirm-Editor, der Ihnen die Eingabe von Text, das Ändern, Speichern und Laden von der Diskette erlaubt. Er läßt Sie auch den ganzen Text oder Teile davon ausdrucken und kann Begriffe finden und durch andere ersetzen. Er basiert auf Intuition, was heißt, daß er die sehr bedienerfreundlichen Features des Amigas, die Ihnen von anderen Programmen her schon vertraut sind, bietet.
Dazu gehören Fenster, Menüs und Mausbedienung. Wenn Sie jedoch die alten Methoden der Computer vor dem Fortschritt des »WIMPs« gewohnt sind, so können Sie auch *GenAm* bedienen, ohne die Maus anfassen zu müssen.
Der Editor ist RAM-orientiert, was bedeutet, daß das ganze File immer im RAM steht und Sie nicht warten müssen, wenn der Editor jeweils Text-Segmente von/zur Disk umspeichert. Weil der Amiga soviel Speicher hat, gibt es ein Speicherlimit, wie man es oft bei älteren Editoren findet, hier nicht. Wenn Sie genug RAM haben, können Sie Files von über 300K editieren (stellen Sie aber sicher, daß dann auf der Diskette noch genug Platz ist, damit Sie auch noch abspeichern können). Weil auch alle Editor-Aktionen, wie die Suchfunktion, RAM-orientiert sind, laufen sie blitzschnell. Ein eingetipptes Programm bringt nicht viel, wenn Sie es nicht auf der Disk sichern können. Deshalb hat der Editor sehr vielfältige Optionen zum Laden und Speichern von Texten, was Ihnen erlaubt, Teile des Textes (oder den ganzen Text) zu sichern, Textteile in die Mitte eines bestehenden Textes einzuladen, um nur einige Beispiele zu nennen.
Für die Bedienung des Editors stehen Ihnen verschiedene Methoden zur Verfügung, die Sie einzeln oder kombiniert anwenden können:

- Steuerung über Tasten, wie die Cursor-Taste oder eine Funktionstaste
- Auswahl eines Menü-Punktes, wie zum Beispiel »Save« per Maus
- Menü-Kürzel, über die Kombination von rechter Amiga-Taste und einem Buchstaben.

Im folgenden wird die Kombination von rechter Amiga-Taste und Taste x mit A-x abgekürzt. Entsprechend wird die Kombination von Alt-Taste und x mit Alt-x bezeichnet.

- Steuerung über die Ctrl-Taste in Kombination mit einem Buchstaben

Im folgenden wird die Kombination von Control-Taste und Taste x mit Ctrl-x abgekürzt.

- Anklicken des Bildschirms, wie zum Beispiel auf der Window-.Gadgets

Die Menü-Kürzel wurden so gewählt, daß sie leicht zu merken sind, die Cursor-Tasten basieren auf dem Amiga-Basic-Editor, während die Control-Codes WordStar-kompatibel sind, wie bei vielen anderen Editoren auch. Wenn Sie mal nicht weiterwissen, bringt ein Druck auf die Help-Taste die Erklärungen aller Tasten, die nicht in einem der Menüs schon sichtbar sind.

2.2.2 Einige Worte über Requester

Der Editor macht extensiven Gebrauch von Requestern, weshalb es nützlich ist zu rekapitulieren, wie man damit umgeht. Die Editor-Requester enthalten nur Text-Gadgets (Gadget = Bedienelement) und Knöpfe.
Text-Gadgets erlauben Ihnen, Text einzugeben und zu editieren. Sie erscheinen als Rechteck, das den Text enthält und einen Block, der die Cursor-Position anzeigt. Sie müssen dieses Rechteck anklicken, bevor Sie irgendein Zeichen eingeben können. Dann können Sie Zeichen eingeben und editieren, wozu die Cursor-Tasten, die Backspace- und die Del-Taste genutzt werden können. Sie können das ganze Feld mit A-x löschen. Sie können den Cursor mit Shift plus Pfeiltaste an den Anfang oder das Ende des Feldes bewegen. Wenn mehr als ein editierbares Text-Gadget in einem Requester ist, können Sie durch Anklicken mit der Maus umschalten.
Knöpfe können mit der Maus angeklickt werden und haben zur Folge, daß der Requester verschwindet. Meistens gibt es einen Vorzugs-Knopf, der durch eine doppelte Einrahmung markiert ist. Wird [Return] gedrückt, hat das dieselbe Wirkung, wenn Sie vorher ein Text-Gadget durch Klicken aktiviert hatten.
In einige Requester können nur bestimmte Zeichen eingegeben werden. So akzeptiert der »`goto line` (Zeile)«-Requester nur Ziffern.

Eingabe von Text und Cursor-Bewegungen

Wenn Sie *GenAmiga* geladen haben, sehen Sie ein leeres Fenster mit einer Statuszeile unten und einem den Cursor darstellenden Block in der oberen, linken Ecke. Die Statuszeile enthält Informationen über die Cursor-Position, (Zeile und Spalte) sowie die Größe des noch freien Textspeichers in Bytes. Zu Anfang wird letztere 59980 zeigen, wenn die voreingestellte Größe von 60000 gewählt wurde. Sie können diesen Voreinstellungswert zusammen mit diversen anderen Optionen ändern, wenn Sie das Programm *GemInst* aufrufen, das detailliert im Kapitel 4 beschrieben ist. Es können auch über Kommandozeile andere Werte übergeben werden (siehe Punkt 2.1). Die » fehlenden« 20 Byte werden vom Editor intern belegt. Der Rest der Statuszeile wird für Fehlermeldungen gebraucht, die normalerweise mit einem Schirmflackern beginnen, um Sie darauf aufmerksam zu machen. Jede Meldung wird entfernt, sobald Sie wieder eine Taste betätigen.
Die Texteingabe erfolgt über die Tastatur. Sobald Sie eine Taste drücken, wird das auf dem Schirm angezeigt und der Cursor um eine Spalte weiterbewegt. Wenn Sie sehr schnell tippen, kann es vorkommen, daß die Anzeige nicht folgen kann. Aber keine Sorge, es geht kein Tastendruck verloren und die Anzeige folgt, sobald Sie eine Pause machen. Am Ende einer Zeile können Sie die Return- oder Enter-Taste benutzen, um eine neue Zeile zu beginnen. Sie können Fehler mit der Backspace-Taste korrigieren, welche das Zeichen links vom Cursor löscht, oder mit der Del-Taste, die das Zeichen auf der Cursorposition löscht.
Der Hauptvorteil eines Computer-Editors gegenüber einer konventionellen Schreibmaschine ist die Fähigkeit, Dinge zu editieren, die Sie lange vorher eingegeben hatten. *GenAmiga* bietet eine Fülle von Optionen, mittels derer Sie sich völlig frei im Text bewegen können.

2.2.3 Cursor-Tasten

Um den Cursor durch den Text zu bewegen, um Fehler zu korrigieren oder neue Zeichen einzugeben, benutzen Sie die vier Pfeiltasten. Wenn Sie den Cursor hinter das rechte Ende einer Zeile bewegen, fügt der Editor deshalb kein Leerzeichen hinzu. Wenn Sie dann ein Zeichen tippen, wird es automatisch an das wirkliche Ende der Zeile gehängt. Wenn Sie lange Zeilen tippen, rollt das Fenster nach rechts, wenn erforderlich.

Wenn Sie mit 'Pfeil nach oben' über die erste Zeile des Windows gehen, wird der Text entweder nach unten rollen (wenn oben noch Text ist) oder es kommt die Meldung »`Top of File`« (Anfang des Textes erreicht). Ähnlich wirkt der Cursor abwärts am Ende des Textes, hier mit der Anzeige »`End of File`«.
Für diejenigen unter Ihnen, die WordStar gewohnt sind: Die Tasten Ctrl-S, Ctrl-D, Ctrl-E, Ctrl-X wirken wie die Cursor-Tasten.
Um direkt auf den Beginn einer Zeile zu gelangen, drücken Sie Alt-Cursor-links, für das Ende der Zeile Alt-Cursor-rechts.
Um den Cursor ein Wort weiter zu bewegen, kombinieren Sie die Tasten Cursor-rechts/links mit der Shift-Taste. Ein Wort ist eine Zeichenkette, abgegrenzt von Leerzeichen, dem Tabulator-Zeichen oder an einem Ende durch Zeilenanfang oder Zeilenende. Sie können auch die WordStar-Codes Ctrl-A und Ctrl-F einsetzen.
Um den Cursor eine Seite aufwärts zu bewegen, nehmen Sie Shift-Cursor-hoch oder Ctrl-R. Eine Seite abwärts gehen Sie mit Shift-Cursor-abwärts oder Ctrl-C.
Wenn Sie den Cursor auf einen beliebigen Platz positionieren wollen, können Sie dafür den Mauszeiger nehmen und an dieser Stelle die linke Maustaste klicken. (Es gibt in WordStar kein Äquivalent für dieses Feature!).

2.2.4 Tabulator-Taste

Die Tabulator-Taste (Tab) schreibt ein spezielles Zeichen (ASCII-Code 9) in den Puffer. Auf dem Schirm erscheint eine Folge von Leerzeichen. Tab justiert den Cursor auf die nächste Spalte, die ein Vielfaches von 8 ist. Genau: Wenn der Cursor am Beginn einer Zeile ist (Spalte 1) und Sie geben Tab, steht der Cursor in Spalte 8+1, also 9. Tabs sind sehr nützlich, um Worte untereinander zu justieren, und ihre Hauptaufgabe in *GenAmiga* ist auch, Befehle untereinanderzuschreiben. Wenn Sie ein Tab löschen, rückt die Zeile auf, als wenn die entsprechende Anzahl Leerstellen gelöscht worden wäre. Der Vorteil der Tabs ist, daß sie nur ein Byte im Speicher belegen, auf dem Schirm aber nach viel mehr aussehen. Sie können die Tab-Weite ändern, bevor oder nachdem Sie *Genam* laden. Um sie vor dem Laden zu ändern, nehmen Sie das *GenInst*-Programm, das in Kapitel 4 beschrieben ist.

2.2.5 Backspace-Taste

Die Backspace-(Rückschritt-) Taste löscht das Zeichen links vom Cursor. Wenn Sie Backspace am Anfang einer Zeile betätigen, löschen Sie das »unsichtbare« CR-Zeichen, womit diese Zeile an das Ende der vorherigen angefügt wird. Backspace am Ende einer Zeile löscht das letzte Zeichen der Zeile, es sei denn, die Zeile ist leer. In diesem Fall wird der Cursor nur nach links gestellt.

2.2.6 Delete-Taste

Die Del-Taste löscht das Zeichen unter dem Cursor und hat keinen Effekt, wenn der Cursor hinter dem Ende einer Zeile steht.

2.2.7 Goto Line

Um den Cursor auf eine bestimmte Zeile (unsichtbare Zeilennummer) zu bewegen, wählen Sie »`goto line`« aus dem Options-Menü oder geben Sie A-G ein. Dann klicken Sie das darauf erscheinende Text-Gadget an und tippen die Zeilennummer gefolgt von [Return] ein. Sie können auch OK anklicken, oder Cancel, wenn Sie den Befehl zurücknehmen wollen. Nach der Auswahl wird der Cursor auf die entsprechende Zeile bewegt, die zugehörige Seite wird abgebildet. Wurde die Zeile nicht gefunden, erscheint die Meldung »`End of file`«.

2.2.8 Goto Top

Um den Cursor auf die erste Zeile des Textes zu bewegen, wählen Sie »Goto Top« aus dem Options-Menü oder geben A-T oder Alt und Cursor aufwärts ein.

2.2.9 Goto Bottom

Um den Cursor auf die letzte Zeile des Textes zu bewegen, wählen Sie »Goto Bottom« aus dem Options-Menü oder geben A-B oder Alt und Cursor abwärts ein.

2.2.10 Verlassen von *GenAm*

Um *GenAmiga* zu verlassen, wählen Sie »Quit« aus dem Project-Menü oder geben Sie A-Q ein. Falls der Text geändert wurde, ohne ihn auf der Disk zu sichern, erscheint ein Alarm-Requester. Klicken Sie dort auf Cancel, sind Sie wieder im Editor, bei OK hingegen verlassen Sie *GenAmiga*, und die Änderung wird nicht gespeichert.

Löschen von Text

2.2.11 Löschen einer Zeile

Die aktuelle Zeile kann mit Ctrl-Y gelöscht werden. Wenn dies die letzte oder vorletzte Zeile betrifft, wird das Fenster neu gezeichnet und zeigt so nur die neue letzte Zeile.

2.2.12 Löschen bis zum Ende einer Zeile

Der Text von der aktuellen Cursor-Position bis zum Ende der Zeile kann mit Ctrl-Q gelöscht werden. (Dies ist äquivalent zu Ctrl-Q-Y in WordStar.)

2.2.13 Gelöschte Zeile wieder einfügen

Wenn eine Zeile mit dem obigen Kommando gelöscht wird, wird sie in einem internen Puffer gesichert und kann mit Ctrl-U wieder eingefügt werden. Dies kann mehrmals wiederholt werden und ist besonders nützlich, wenn Sie für viele ähnliche Zeilen eine Zeile auf diese Art schnell kopieren wollen.

2.2.14 Gesamten Text löschen

Wenn Sie den ganzen Text löschen wollen, wählen Sie »Clear« aus dem Project-Menü oder geben A-C ein. Wenn Sie irgend etwas im Text geändert haben, was noch nicht auf der Disk gesichert wurde, wird mittels eines Requesters Bestätigung verlangt. Klicken Sie OK, wenn Sie den Text löschen wollen, sonst Cancel.

Disketten-Operationen

Es ist nutzlos, wenn Sie nur Text eintippen, ohne ihn danach wieder abspeichern oder laden zu können. Daher besitzt der Editor eine Menge Features, um von der Disk zu lesen oder auf sie schreiben zu können.

2.2.15 Text sichern

Save as (Speichern als)

Um den Text, den Sie gerade editieren, zu sichern, wählen Sie »Save as« aus dem Project-Menü oder geben A-S ein. Es erscheint ein Requester, der Ihnen die Eingabe eines passenden File-Namens erlaubt. Wenn Sie OK anklicken oder [Return] drücken, wird das File auf der Disk gesichert. Falls dabei ein Fehler auftritt, erscheint ein Requester mit einer Meldung oder einer Amiga-DOS-Fehlernummer. Deren Bedeutung finden Sie im Anhang A. Wenn schon ein File gleichen Namens existiert, wird es gelöscht und durch den neuen Inhalt ersetzt. Wenn Sie im Requester »Cancel« anklicken, wird das File nicht gesichert.

Save (Speichern)

Wenn Sie bereits ein »Save as« (oder ein Load) ausgeführt haben, merkt sich *GenAmiga* den File-Namen und zeigt ihn als Fenster-Titel an. Mit «Save» aus dem Project-Menü wird der alte Name einfach wieder eingesetzt. Wenn Sie versuchen, mit »Save« zu sichern, ohne vorher einen File-Namen definiert zu haben, erscheint der Requester, wie unter »Save as«.

2.2.16 Text laden

Um ein neues Text-File zu laden, wählen Sie »Load« aus dem Project-Menü oder geben A-L ein. Falls Sie das vorhandene File geändert hatten, ohne zu sichern, muß dies bestätigt werden. Der erscheinende File-Requester ermöglicht die Eingabe des File-Namens. Angenommen, Sie haben nicht »Cancel« gewählt, wird der Editor versuchen, das File zu laden. Ist das File zu groß, erscheint die Meldung »`Not enough memory`« (nicht genug Speicher), und das Fenster blinkt kurz auf. Anmerkung des Übersetzers: Sie sollten dann, wie bereits geschildert, den Editor-Puffer größer wählen.

2.2.17 Text-File einfügen

Wenn Sie ein Text-File ab aktueller Cursor-Position einfügen wollen, wählen Sie »Insert« aus dem Project-Menü oder geben A-I ein. Es erscheint der übliche File-Requester. Wenn Sie nicht »Cancel« wählen, wird das File eingelesen, vorausgesetzt, es paßt noch in den Speicher.

2.2.18 Suchen und Ersetzen

Um einen bestimmten Text zu finden, wählen Sie »Find« aus dem Search-Menü oder geben A-F ein. Es wird ein Requester erscheinen, der die Eingabe des Such-Textes und – wenn gewünscht – des Ersetz-Textes ermöglicht. Wenn Sie »Cancel« anklicken, wird die Funktion abgebrochen. Ein Klick auf »Next« startet die Suche vorwärts, ein Klick auf »Previous« die Suche rückwärts. Wenn Sie nicht wollen, daß der gefundene Text durch einen anderen ersetzt wird, lassen Sie das Ersetz-Feld leer. Wenn die Suche erfolgreich war, wird das Fenster neu gezeichnet und der Cursor auf den Beginn des Suchbegriffs gesetzt. Wenn der Text nicht gefunden werden konnte, erscheint die Meldung »`Not found`« (nicht gefunden) in der Statuszeile, und der Cursor bleibt da, wo er war. Bei der Suche sind Groß- und Kleinschreibung gleichwertig. Sie können also `Test`, `test` oder `TEST` eingeben.
Um das nächste Auftreten des Suchtextes zu finden, wählen Sie »Find next« aus dem Search-Menü oder geben A-N ein. Die Suche beginnt an der Position direkt nach dem Cursor.
Um das vorherige Auftreten des Suchtextes zu finden, wählen Sie »Find previous« aus dem Search-Menü oder geben A-P ein.Die Suche beginnt an der Position direkt vor dem Cursor.
Wurde ein Suchtext gefunden, kann er durch den Ersetz-Text ersetzt werden, indem man »Replace« im Search-Menü anklickt oder A-R eingibt. Nach dem Ersetzen sucht *GenAmiga* automatisch nach dem nächsten Auftreten des Suchtextes.
Wenn Sie den Suchtext jedesmal ab Cursorposition vorwärts ersetzen wollen, so wählen Sie »Replace all« aus dem Search-Menü.
Das Menü ist absichtlich so gestaltet, daß ein versehentliches globales Ersetzen verhindert wird. Aus diesem Grund gibt es dafür auch kein Tastatur-Kürzel. Während des Vorgangs können Sie mit der Escape-Taste abbrechen.

Block-Kommandos

Ein Block ist ein markierter Textbereich, der kopiert, gelöscht, gedruckt oder auf Disk geschrieben werden kann. Dafür werden die Funktionstasten benutzt.

2.2.19 Markieren eines Blocks

Der Beginn eines Blocks wird markiert, indem man den Cursor auf die entsprechende Stelle bringt und F1 drückt. Das Ende wird markiert, indem man den Cursor dorthin bewegt und F2 drückt. Beginn und Ende eines Blocks müssen nicht in dieser Reihenfolge markiert werden.

2.2.20 Sichern eines Blocks

Wenn ein Block markiert ist, kann er mit F3 gesichert werden. Wenn kein Block markiert ist, erscheint die Meldung »`What blocks!`«. Liegt der Beginn eines Blocks in der Textreihenfolge hinter dem Ende, erscheint die Meldung »`Invalid block!`«. Beide Fehler beenden das Kommando.
Wenn ein gültiger Block markiert ist, erscheint ein File-Requester für die Eingabe eines passenden File-Namens. Wenn Sie einen Namen wählen, der schon existiert, wird das vorhandene File überschrieben.

2.2.21 Kopieren eines Blocks

Ein markierter Block kann auf eine andere Textstelle kopiert werden, indem man den Cursor an die Stelle bewegt und F4 drückt. Wenn Sie versuchen, einen Block in sich selbst zu kopieren, erscheint die Meldung »Invalid block«, und der Vorgang wird abgebrochen.

2.2.22 Löschen eines Blocks

Ein markierter Block kann mit Shift und F3 gelöscht werden. Die Shift-Taste ist zusätzlich erforderlich, um ein versehentliches Löschen zu vermeiden.

2.2.23 Drucken eines Blocks

Ein markierter Block kann ausgedruckt werden, indem man »Print Block« aus dem Project-Menü wählt oder A-W eingibt. Es wird ein Requester erscheinen, der nach dem Drucker fragt. »PRT:« ist schon voreingestellt. Ein Klick auf OK startet den Ausdruck. Natürlich können Sie anstatt »PRT:« auch jedes andere gültige Amiga-Gerät angeben, so daß Sie auch den Block in ein Disk-File »drucken« können. Es besteht dabei aber ein Unterschied zum »Save Block«, wo die Tab-Zeichen durch die entsprechende Anzahl Blanks ersetzt werden.
Wenn kein Block markiert ist, wird der ganze Text gedruckt.
Blockmarkierungen bleiben über alle Editier-Kommandos bestehen und wandern automatisch mit, wenn nötig. Sie können nur durch die Kommandos «Clear», «Delete block» und «Load» aufgehoben werden.
Wenn Sie eine Zeile editieren, die eine Blockmarkierung enthält, ist die neue Position der Marker undefiniert.

Verschiedenes

2.2.24 Automatisches Einrücken

Es dient der Übersichtlichkeit beim Editieren von Programmen – sowohl in Assembler als auch in Hochsprachen – Folgezeilen links einzurücken. Der Editor unterstützt dieses automatische Einrücken, das an- und abgeschaltet wird, indem man »Autoindent on« im Options-Menü anklickt. Nach dem Laden ist es immer ausgeschaltet. Wenn es aktiviert wird, wird nach jedem Return zu Beginn der neuen Zeile soviel an Leerraum (Tabs und Blanks) eingefügt, wie die vorherige hatte.

2.2.25 Tabulatoren ändern

Die Voreinstellung für den Tabulator ist 8, kann aber während des Editierens durch die Wahl von »Set Tabs« aus dem Options-Menü geändert werden. Es erscheint ein Requester, der die aktuelle Einstellung zeigt. Diese kann auf einen Wert zwischen 2 und 16 geändert werden.

Wenn Sie die Voreinstellung dauerhaft ändern wollen, die *GenAmiga* nach jedem Laden vorgibt, nehmen Sie dazu das *GenInst*-Programm, das in Kapitel 4 beschrieben ist. Beachten Sie jedoch, daß einige Programme und die meisten Drucker einen 8er-Tabulator voraussetzen.

2.2.26 Hilfe-Schirm

Die Tastatur-Befehle, die nicht in den Menüs aufgeführt sind, werden angezeigt, wenn Sie die Help-Taste oder A-H drücken. Es erscheint ein Requester, der sowohl die Tasten erklärt, als auch den noch freien Speicher anzeigt.

2.2.27 Assemblieren

Um den Assembler aufzurufen, wählen Sie »Assemble« aus dem Options-Menü oder geben A-A ein. Dies wird später noch genauer erläutert.

2.2.28 Anspringen der Fehlerzeile

Während des Assemblierens werden die ersten 30 Fehler und ihre Zeilennummern in einem Puffer gespeichert. Wenn Sie zum Editor zurückkehren, können Sie diese Zeilen schrittweise abarbeiten (ändern), indem Sie »Jump to error« aus dem Options-Menü wählen oder A-J geben. Dies bewegt den Cursor auf die nächste Zeile, für die ein Fehler gemerkt wurde, und zeigt die Fehlermeldung in der Status-Zeile.
Wenn Sie irgendwelche Änderungen vornehmen, die die Anzahl der Textzeilen ändern, wird der Fehler-Puffer gelöscht und die Meldung »Errors lost« erscheint. Wenn es keine Fehler mehr gibt, erscheint die Meldung »No more errors«.

2.2.29 Das Editor-Fenster

Das Fenster, das der Editor nutzt, funktioniert wie die meisten anderen Amiga-Fenster. So können Sie es verschieben mit dem Verschiebe-Gadget oben. Sie können seine Größe mit dem Größen-Gadget links unten ändern. Sie können seine Lage relativ zu den anderen Fenstern ändern, indem Sie auf das Front- beziehungsweise Back-Gadget rechts oben klicken.
Wenn Sie das Schließ-Gadget links oben anklicken, hat das denselben Effekt, als wenn Sie »Quit« aus dem Project-Menü wählen oder A-Q geben.
Wie alle Intuition-Fenster muß auch dieses vor einer Eingabe oder Menü-Wahl aktiviert werden (irgendwo im Fenster klicken).

Wenn Sie einen größeren Schirmbereich als normal haben, wie zum Beispiel bei den europäischen PAL-Versionen (256 anstatt 200 Zeilen), können Sie das Fenster auf die volle Schirmgröße ziehen.
Wenn *GenAmiga* während eines Diskzugriffs oder während des Assemblierens beschäftigt ist, erscheint anstelle des Mauszeigers eine Sanduhr.

2.3 Der Makro-Assembler

2.3.1 Einführung

GenAmiga ist ein mächtiger, schneller Assembler, der mit dem Editor gekoppelt ist. Er wandelt Text, der in den Editor eingetippt oder geladen wurde, zusammen mit optional von der Disk nachladbaren Texten in einen Binär-File um, wahlweise als linkbaren oder sofort ausführbaren Code.
GenAmiga ist ein Zwei-Pass-Assembler (arbeitet in zwei Durchläufen). Im ersten Durchlauf untersucht er den gesamten Speichertext und ggf. die Disk-Files und baut daraus die Symboltabelle auf. Während des zweiten Durchlaufs erfolgt die Umsetzung der Befehle in Bytes (68000-Code); ein Listing und ein Binär-File können dabei generiert werden.
Im ersten Durchlauf werden nur fatale Fehler gemeldet (zum Beispiel »`Include-File nicht gefunden`«), die sofort zum Abbruch des Assembler-Laufs führen. Während Durchlauf zwei werden alle Fehler angezeigt, wenn erforderlich zusammen mit dem Listing und der Symboltabelle. Bevor wir auf den Assembler selbst näher eingehen, sollen die unterschiedlichen Code-File-Typen und die zugehörigen Erstellungs-Optionen beschrieben werden.

2.3.2 Binäre File-Typen

Es gibt zwei binäre Filetypen, die mit *GenAmiga* erzeugt werden können.

Ausführbare Files

GenAmiga produziert normalerweise direkt ausführbare Files, die sofort als Programm gestartet werden können. Die Notwendigkeit für einen extra Linker-Lauf (wie beim Standard-Assembler) besteht nicht. Ausführbare Files haben üblicherweise keinen Extender.

Linkbare Files

Diese können nicht sofort gestartet werden, sondern müssen in den Standard-Amiga-Linker ALink oder den schnelleren Linker BLink (Public Domain und auf der *Devpac*-Diskette) eingelesen und mit anderen Modulen gebunden

werden. Für linkbare Files werden die Extender ».o« oder ».obj« empfohlen. Per Voreinstellung produziert *GenAmiga* ausführbare Files. Die können aber mittels einer Direktive (Option l+) in »linkbar« geändert werden, auch die Änderung der Voreinstellung ist mittels *GenInst* möglich.

2.3.3 Codearten

Wenn man den 68000 programmiert, sind zwei »Programmiermodi« möglich. *GenAmiga* unterstützt sowohl den positionsunabhängigen, als auch den positionsabhängigen Code.

Positionsunabhängiger Code

Mit dem 68000 ist es sehr viel einfacher, diese Codearten zu programmieren, als mit älteren Prozessoren. Die PC-relative Adressierung und die relativen Sprunganweisungen stellen eine große Erleichterung dar, aber es gibt ein Problem, wenn man Daten an einer bestimmten Stelle (in einer Variablen) ablegen will. Nehmen Sie zum Beispiel an, Sie möchten den Wert 1 in »where« speichern, dann geht das sicherlich nicht:

```
move.w     #1,where(pc)        ;falsch!
```

PC-relative Adressierung kann nicht auf den Zieloperanden angewandt werden. Eine Lösung des Problems wäre:

```
lea        where(pc),a0
move.w     #1,(a0)
```

Ein anderer Weg ist es, ein Adreßregister als Zeiger auf den RAM-Bereich zu setzen, in dem die Variablen gehalten werden, und dann über den Index auf einzelne Variable zuzugreifen. Nehmen wir an, »datast« ist Adresse 6 Byte vor »where« und Adreßregister A6 wurde zu Programmbeginn auf »datast« gestellt, so gilt:

```
move.w     #1,6(a6)
```

Es gibt eine spezielle Direktive in *GenAmiga*, diese Dinge zu vereinfachen (die RS-Direktive). *GenAmiga* selbst wurde unter Anwendung dieser Methode geschrieben.

Positionsabhängiger Code

Dank des Amiga-DOS-Programmladers kann man auf dem Amiga auch positionsabhängige Programme laufen lassen. Sie wissen allerdings nie, wohin im Speicher Ihr Programm geladen wird, jedenfalls so lange nicht, bis es aus-

geführt wird (und Sie es dann selbst feststellen). Mit *GenAmiga* schreiben Sie einfach Ihr (positionsabhängiges) Programm und der Assembler packt dazu die passende Relokatiertabelle an das Ende des Files. Mit Hilfe dieser Tabelle berechnet und ersetzt der Amiga-DOS-Lader die absoluten Adressen. Das genannte Problem kann also schlicht gelöst werden mit

```
move.w   #1,where
```

Der Nachteil des positionsabhängigen Codes ist, daß er generell mehr Speicher verbraucht, weil das Binär-File um die Länge der Relokatiertabelle größer ist. Andererseits ist der Quelltext im allgemeinen einfacher zu lesen. In den Assembler-Listings beginnt der Programmzähler normalerweise bei null, und jede Adresse sollte relativ zum Programmstart gesehen werden.

2.3.4 Starten des Assemblers

Der Assembler wird vom Editor aus gestartet, indem man »Assemble« aus dem Options-Menü wählt oder A-A gibt. Es erscheint dann ein Requester, der die Eingabe des Binär-File-Namens und des Listing-File-Namens erlaubt. Sie können beliebige Namen eingeben, auch solche mit Extendern. Wenn Sie eines oder beide Felder leerlassen, wird kein Binär und/oder kein Listing-File erzeugt. Wenn Sie als Listing-Namen den Stern (*) eingeben, geht das Listing auf den Bildschirm (in das *GenAm*-Window). Wenn Sie »Cancel« anklicken, wird die Operation abgebrochen, mit OK wird sie gestartet.
Zuerst prüft *GenAmiga*, ob es das/die Binär/Listing-Files öffnen kann. Ein existierendes File gleichen Namens wird überschrieben. Tritt ein Fehler auf, erscheint ein Requester mit der Fehlermeldung, andernfalls beginnt das Assemblieren.
Die Meldung »`Pass 1`« erscheint auf dem Bildschirm, danach die Meldung »`Pass 2`«. Während Pass 2 wird ein Listing generiert, sofern dies vorgewählt wurde. Zusätzlich erscheinen Fehlermeldungen und Warnungen (auch im Listing). Wenn kein Listing angefordert wurde, werden nur eventuelle Fehlermeldungen ausgegeben. Nach dem Ende von Pass 2 erscheinen diverse Meldungen über die Anzahl der Fehler, die Anzahl der assemblierten Zeilen, den Codeart und die Auslastung des Raums für Symbol- und Makro-Tabelle. Falls ein Listing in irgendeiner Form verlangt war, wird auch die Symbol-Tabelle ausgegeben (falls dies nicht mittels der Opt-S-Direktive unterdrückt wurde).
Anmerkung des Übersetzers: Wenn Sie die mitgelieferten Include-Files einsetzen, sollten Sie via Opt-S die Symbol-Ausgabe unterdrücken. Andernfalls entstehen zum Teil sehr lange Tabellen, die bei Ausgabe in ein Disk-File bei mir schon Disketten »gesprengt« haben.

Nach dem Assemblieren gelangen Sie durch Druck auf eine beliebige Taste wieder in den Editor.
Während des Assemblierens kann die Bildschirmausgabe durch Druck auf eine beliebige Taste angehalten und mit [Return] wieder fortgesetzt werden. Ansonsten kann das Assemblieren jederzeit mit der Escape-Taste oder durch Anklicken des Close-Gadgets im Editor-Fenster abgebrochen werden. Das dabei entstandene Code-File ist natürlich unvollständig und ungültig und sollte nicht ausgeführt werden.
Wenn Sie mehrmals hintereinander assemblieren, werden Sie feststellen, daß sich der Assembler die Filenamen gemerkt hat, die Sie vorher eingegeben hatten.
Nun zur Syntax des Assemblers.

2.3.5 Assembler-Befehls-Format

Jede Zeile, die der Assembler verarbeiten soll, sollte dieses Format haben:

Label-Feld	Mnemonik-Feld	Operanden-Feld	Kommentar-Feld
start	move.l	d0,(a0)+	Ergebnis speichern

Ausgenommen von dieser Regel sind Kommentarzeilen, die mit einem Stern (*) gekennzeichnet sind, und Leerzeilen, die ignoriert werden. Jedes Feld muß vom nächsten durch einen Zwischenraum getrennt sein, wofür jede Kombination von Leer- und Tab-Zeichen erlaubt ist.

Das Label-Feld (Markenfeld)

Ein Label (Marke) beginnt normalerweise in Spalte 1, aber wenn es nötig erscheint, es in einer anderen Spalte beginnen zu lassen, muß im direkten Anschluß ein Doppelpunkt (:) folgen. Labels sind in allen Zeilen mit Befehlen erlaubt, aber bei einigen Assembler-Direktiven verboten, bei anderen wiederum sind sie Pflicht.
Alle Labels müssen mit einem der Zeichen a..z, A..Z, Umlauten/ß oder dem Unterstrich (_) beginnen. Danach sind noch die Zeichen 0..9 und der Punkt erlaubt. Makro-Namen und Register-Equates dürfen keinen Punkt enthalten.
Kleinbuchstaben werden normalerweise von den entsprechenden Großbuchstaben unterschieden, solange dies nicht über die Opt-C-Direktive oder im *GenInst*-Programm geändert wird. Die ersten 16 Zeichen eines Labels sind kennzeichnend. Labels dürfen nicht die gleichen Namen wie Register haben oder reservierte Wörter wie SR, CCR oder USP beinhalten.

Einige Beispiele für korrekte Labels:

```
test, Test, TEST, _test, _test.end, test5, _5test
```

Beispiele für inkorrekte Labels:

```
.test, 5test, _&e, test>
```

Das Mnemonik-Feld

Das Mnemonik-Feld folgt dem Label-Feld und kann 68000-Befehle, Assembler-Direktiven oder Makro-Aufrufe enthalten. Einige Befehle und Direktiven erlauben die Typ-Angabe mittels .B für Bytem, .W für Wort und .L für Langwort, sowie .S für Short. Was wann erlaubt ist, hängt vom jeweiligen Befehl beziehungsweise von der Direktive ab.
GenAmiga ignoriert die Groß/Kleinschrift-Unterscheidung bei Befehlen und Direktiven, weshalb move und MOVE und mOve gleichwertig sind.

Das Operanden-Feld

Für Befehle und Direktiven, die Operanden erfordern, enthält dieses Feld ein oder mehrere durch Kommas getrennte Parameter. Registernamen können groß oder klein geschrieben werden, das heißt, SP und sp sind gleich und gültige Namen für Stackpointer.

Das Kommentar-Feld

Jedes Leer- oder Tab-Zeichen, das nach dem zu erwartenden Ende von Operand(en) gefunden wird, wird als Beginn des Kommentar-Feldes betrachtet, das der Assembler ignoriert.
Hinweis des Übersetzers: Vermeiden Sie unbedingt Leerstellen zwischen den Operanden, weil der Assembler dies als Kommentarbeginn auffaßt. Speziell bei Auflistungen nach »DC« kann das problematisch werden. Mindestens ein Leerzeichen (oder Tab) ist obligatorisch.

`rts ;`	das ist OK
`rts;`	das ist falsch

Beispiel für gültige Zeilen

```
move.l      d0,(a0)+            Kommentar hier
loop TST.W      d0
nur.label
rts
* das ist eine reine Kommentar-Zeile
eingerückt:     link A6,#-10    schaffe Platz
a_string: dc.b 'Leerstellen in Hochkommas erlaubt'    Ein String
```

2.3.6 Ausdrücke

GenAmiga erlaubt komplexe Ausdrücke und unterstützt voll die Operatoren-Priorität, Klammern und logische Operatoren.
Es gibt zwei Arten von Ausdrücken – absolute und relative – und ihre Unterscheidung ist wichtig. Absolute Ausdrücke sind Konstanten, die dem Assembler bekannt sind. Relative Ausdrücke sind (ergeben) Programm-Adressen, die während des Assemblierens nicht bekannt sind, weil das Programm vom AmigaDOS-Lader an eine beliebige Stelle im Speicher verlagert werden kann. Für einige Befehle und Direktiven gibt es Einschränkungen hinsichtlich des erlaubten Typs, und einige Operatoren können nicht mit bestimmten Typ-Kombinationen benutzt werden.

Operatoren

Die zur Verfügung stehenden Operatoren in absteigender Priorität sind:

minus(–) und plus (+) als Vorzeichen
Bitweise NOT (~)
Links («) und rechts(») schieben
Gleichheit (=)
Bitweises AND (&), OR (!) und XOR(^)
Multiplikation(*) und Division(/)
Addition(+) und Subtraktion(–)

Die Priorität kann durch die Klammern () aufgehoben werden. Ausdrücke mit Operatoren gleicher Priorität werden von links nach rechts entwickelt. Leerstellen in Ausdrücken (ausgenommen in Strings in Hochkommas) sind nicht erlaubt, weil sie als Trennzeichen zum nächsten Feld aufgefaßt werden.
Alle Ausdrücke werden in 32-Bit-Arithmetik mit Vorzeichen gerechnet, es findet kein Test auf Überlauf statt.

Zahlen

Absolute Zahlen sind in verschiedenen Darstellungen möglich als

Dezimale Konstanten, z.B. 1029
Hexadezimale Konstanten, z.B. $12f
Binäre Konstanten, z.B. %1000101
Zeichen-Konstanten, z.B. 'X'

$ bezeichnet hexadezimale Zahlen, % binäre und ' oder " Zeichen-Konstanten.

Zeichen-Konstanten

Ob Anführungszeichen oder Hochkommas zur Stringabgrenzung benutzt werden ist unbedeutend, jedoch müssen die beiden Zeichen zu Beginn und Ende des Strings die gleichen sein. Zeichen-Konstanten können bis zu 4 Zeichen lang sein und werden rechtsbündig und mit führenden Nullen dargestellt, wenn erforderlich. Hier einige Beispiele mit den zugehörigen ASCII- und Hexa-Äquivalenten:

```
"Q"      Q     $00000051
'hi'     hi    $00006869
"Test"   Test  $54637374
"it's"   it's  $6974277C
'it''s   it's  $6974277C
```

Strings in DC.B-Direktiven unterliegen leicht unterschiedlichen Regeln, was später noch gezeigt wird.

Erlaubte Typ-Kombinationen

Symbole in Ausdrücken werden entweder relativ oder absolut »entwickelt«, was davon abhängt, wie sie definiert wurden. Labels innerhalb des Programmbereichs (zu Befehlen) sind relativ, während solche auf EQU-Direktiven den Typ des zugewiesenen Wertes annehmen.
Der Stern (*) als Operand bezeichnet den aktuellen Wert des PC am Beginn eines Befehls oder einer Direktive und ist immer relativ.
Die folgende Tabelle faßt zusammen, welche Typ-Kombinationen zu welchem Resultat führen.

Es bedeuten R Relativ
A Absolut
* Kombination ist nicht erlaubt, erzeugt Fehlermeldung.

op mix	A op A	A op R	R op A	R op R
Positives/negatives Vorzeichen	A	*	*	*
Gleichheit	A	*	*	A
Bitweise Operatoren	A	*	*	*
Multiplikation	A	*	*	*
Division	A	*		*
Addition	A	R	R	*
Subtraktion	A	*	R	A

2.3.7 Adressierungsarten

Die verfügbaren Adressierungsarten zeigt die folgende Tabelle. Bitte beachten Sie, daß *GenAmiga* beim Abarbeiten der Adressierungsarten Groß-Klein-Buchstaben nicht unterscheidet, weshalb zum Beispiel sowohl D0 und als auch a3 gültige Registernamen sind.

Form	Bedeutung	Beispiel
Dn	Datenregister direkt	D3
An	Adreßregister direkt	A5
(An)	ARI	(A1)
(An)+	ARI mit Postinkrement	(A5)+
-(An)	ARI mit Prädekrement	-(A0)
d(An)	ARI mit Offset	20(A7)
d(An,Rn.s)	ARI mit Index	4(A6,D4.L)
d.W	Absolut kurz	$0410.W
d.L	Absolut lang	$12000.L
d(PC)	PC-relativ mit Offset	NEXT(PC)
d(PC,Rn.s)	PC-relativ mit Offset und Index	NEXT(PC,A2.W)
#d	Konstante	
Hierin bedeuten:		
ARI:	Adreßregister indirekt :	
n:	Eine Zahl von 0 bis 7	
d:	Eine Zahl	
R:	Indexregister A oder D	
s:	Typ W oder L, W wenn nicht angegeben	

Spezielle Adressierungsarten

CCR Condition Code Register (User-Byte des Statusregisters)
SR Status-Register
USP User Stack Pointer
Zusätzlich kann SP anstatt A7 in allen o.g. Adressierungsarten eingesetzt werden, z.B. 4(SP.D3.W).
Außerdem kann »Adreßregister indirekt« ohne Offset geschrieben werden, somit sind gleichbedeutend (A3,D3.W) und 0(A3,D3.W).

2.3.8 Symbole und Punkte

Symbole, die das Punkt-Zeichen enthalten, können bei *GenAmiga* zu Problemen bei der absolut kurzen Adressierung und bei Register-Equates führen. Der Motorola-Standard zur Kennzeichnung der Kurzadressierung verursacht Probleme, weil Punkte auch in einem Label auftreten können, wie ein Beispiel am besten zeigt:

```
move.w    vector.w,d0
```

Dabei ist »vector« ein absoluter Wert, wie zum Beispiel eine System-Variable. Dies erzeugt die Fehlermeldung »undefined label«, weil ein Label erwartet wird.
Eine Lösung wäre, den Backslash (\) anstatt des Punktes einzusetzen, z.B.

```
move.w    vector\w,d0
```

Hingegen kann der Punkt immer noch in numerischen Ausdrücken gebraucht werden. z.B.

```
move.w    $4.w,d0
```

Alternativ kann der Ausdruck in Klammern gesetzt werden, z.B.

```
move.w    (vector.w),d0
```

Es gibt ein ähnliches Problem, wenn Sie Register-Equates als ein Index-Register einsetzen und versuchen, den Typ anzugeben, z.B.

```
move.b    d0, 0 (a6.buf.l)
```

wobei »buf« vorher einem Register zugewiesen wurde. Dies wird von *GenAmiga* korrekt assembliert, weil Punkte in Namen für Register-Equates verboten sind.

Assembler-Direktiven

GenAmiga kennt bestimmte Pseudo-Mnemoniks. Diese Assembler-Direktiven, wie sie auch genannt werden, werden (normalerweise) nicht in 68000-Code übersetzt, sondern veranlassen den Assembler zu bestimmten Aktionen während des Assemblerlaufs. Diese Aktionen bewirken Unterschiedliches, wie die Änderung des erstellten Objekt-Codes oder der Form des Listings. Direktiven werden genauso wie Befehle behandelt, können auch Labels haben, (einige müssen), und können von einem Kommentar gefolgt sein. Wenn Sie ein Label vor eine Direktive setzen, die das nicht erfordert, ist deren Wert undefiniert und wird normalerweise als Label ignoriert.
Im folgenden werden alle Direktiven nacheinander beschrieben. Bitte beachten Sie, daß die Schreibweise (groß/klein) unwichtig ist, obwohl hier die Großschreibung verwendet wurde. Spitze Klammern (< >) bezeichnen optionale Parameter.

2.3.9 Assembler-Kontrolle

Die folgenden Direktiven beeinflussen den Assemblerlauf.

2.3.10 END

END bezeichnet das Textende für den Assembler, alles was danach kommt wird ignoriert. Es sollte in der letzten Zeile stehen, ist aber nicht Pflicht.

2.3.11 INCLUDE filename

Diese mächtige Direktive holt ein Stück Quelltext von der Disk. Er wird genauso behandelt, als ob er im RAM wäre. Der Direktive muß ein File-Name im gültigen Amiga-Format folgen. Enthält der Name Leerstellen, muß er in Anführungszeichen oder Hochkommas stehen. Laufwerks- und Directory-Angaben sind möglich, z.B.

```
include   df1:constants/header.s
```

Include-Direktiven können bis zu einer Tiefe von 8 geschachtelt sein. Tritt ein Fehler beim Öffnen oder Lesen des Files auf, wird mit einer Fehlermeldung abgebrochen.

2.3.12 INCDIR "directory" <,"directory" usw.>

Diese Direktive gibt eine Liste von Directories an, in denen (und in dieser Reihenfolge) Include-Direktiven die Files suchen sollen. *GenAmiga* setzt dafür die Strings der Reihe nach vor die Filenamen der Include-Direktiven. Die INCDIR-Direktive muß vor irgendeiner Include-Direktive im Text stehen.

2.3.13 OPT option <,option,option, usw.>

OPT erlaubt die Kontrolle diverser Optionen innerhalb von *GenAmiga*. Jede davon wird durch einen Buchstaben – gefolgt von einem Plus- oder Minus-Zeichen – bezeichnet. Mehrere Optionen können durch Kommas getrennt angegeben werden.

2.3.14 Optionen

Die folgenden 8 Optionen (siehe auch OPT) sind erlaubt:

Option C – Case-Unterscheidung bei Labels

»case-sensitiv« heißt, daß zwischen Groß- und Kleinbuchstaben unterschieden wird. Per Voreinstellung ist *GenAmiga* case-sensitiv, was durch C– (nicht case-sensitiv) überschrieben oder mit C+ wieder zurückgesetzt werden kann. Eine andere Voreinstellung ist mit *GenInst* möglich.
Die Direktive muß im Text stehen, bevor irgendein Label eingesetzt oder definiert wird. Beachten Sie bitte, daß die Voreinstellung konträr zu den anderen 68000er *Devpacs* (Atari ST oder Ql) steht.

Option D – Debugging Information

Das Amiga-DOS-Format für Binär-Files erlaubt es, eine Symbol-Tabelle an das File-Ende zu hängen, die von Debuggern wie *MonAmiga* gelesen werden kann und beim Debuggen sehr nützlich ist.
Per Voreinstellung ist diese Option nicht aktiv, kann aber mit D+ ein- bzw. mit D– ausgeschaltet werden. Es werden nur relative Labels aufgezeichnet und zwar in Großschrift, wenn *GenAmiga* nicht auf case-sensitiv eingestellt ist. Wenn mehr als eine D-Option im Text auftritt, wirkt nur die letzte.

Option L – Linker Modus

Normalerweise erstellt *GenAmiga* ausführbaren Code (falls nicht mit *GenInst* geändert), L+ generiert linkbaren Code, und L– stellt wieder ausführbaren Code ein. Die Direktive muß in der ersten Zeile des Textes stehen (auch keine Leerzeile davor!). Wenn der Linker-Modus aktiv ist, sind noch zusätzliche Direktiven möglich, was an späterer Stelle gezeigt wird.

Option M – Makro-Expansion

Wenn ein Assembler-Listing erzeugt wird, werden darin enthaltene Makro-Aufrufe wie im Quelltext ausgegeben. Wenn Sie wollen, daß die Befehle innerhalb der Makros gelistet werden, wählen Sie M+. M– schaltet diese Option wieder aus. Diese Direktive können Sie in einem Text so oft einsetzen wie erforderlich.

Option N – Narrow (schmales) Listing

Die Option ist für Drucker mit nicht mehr als 80 Spalten vorgesehen. Wenn Sie mit N+ aktiviert ist, wird ein verkürztes Listing ohne Zeilennummern und ohne Objekt-Code erzeugt. Zum Umschalten auf »breit« dient N–.

Option P – Positionsunabhängiger Code

Normalerweise erlaubt *GenAmiga* positionsunabhängigen Code. Wenn Sie aber nur PC-relativen Code generieren wollen, so können Sie die Option P+ als Kontrolle einsetzen. Dies erzeugt dann Fehlermeldungen, wenn der Code nicht positionsunabhängig ist. Mit P– wird dies wieder ausgeschaltet. Mehrere P-Optionen in einem Text sind möglich, um zum Beispiel sicherzustellen, daß ein bestimmter Programmteil positionsunabhängig ist.

Option S – Symbol-Tabelle

Wenn ein Listing erzeugt wird, gehört dazu eine Symbol-Tabelle am Ende des Listings. Wenn Sie diese nicht benötigen, können Sie sie mit S– ausschalten bzw. mit S+ wieder einschalten.Nur die letzte Anweisung gilt.

Anmerkung des Übersetzers: Die Symbol-Tabelle der Include-Files kann sehr lang werden; es ist ratsam, sie abzuschalten.

Option W – Warnungen

Um die Warnmeldungen von *GenAmiga* abzuschalten, wählen Sie W–, W+ aktiviert sie wieder. Diese Direktive kann beliebig oft angewählt werden.

Zusammenfassung der Optionen

C+	Case-sensitiv (Voreingestellt, wenn nicht anders installiert)
C–	Nicht case-sensitiv
D+	Debbuger-Information → Binär-File
D–	Keine Debbuger-Information → Binär-File (Voreinstellung)
L+	Erzeugt linkbaren Code
L–	Erzeugt ausführbaren Code (Voreinstellung)
M+	Expandiert Makros im Listing
M–	Keine Makro-Expansion im Listing (Voreinstellung)
N+	Schmales Listing
N–	Breites Listing (Voreinstellung)
P+	Positionsunabhängiger Code
P–	Positionsabhängiger Code (Voreinstellung)
S+	Symboltabelle mit Listing (durch LIST gesetzt)
S–	Keine Symboltabelle (durch NOLIST gesetzt)
W+	Warnmeldungen ausgeben
W–	Warnmeldungen unterdrücken.

Zum Beispiel die Zeile

```
opt  m+,s+,w-
```

schaltet die Makro-Expansion im Listing ein, veranlaßt die Ausgabe der Symboltabelle und unterdrückt Warnmeldungen.

2.3.15 EVEN

Diese Direktive setzt den Programmzähler auf eine gerade Adresse, d.h. er wird auf eine Wortgrenze justiert. Weil *GenAmiga* automatisch alle Befehle auf eine Wortgrenze justiert (ausgenommen DC.B und DS.B), ist EVEN nicht besonders oft erforderlich. Es kann aber sehr nützlich sein, um sicherzustellen, daß Strings und Puffer auf einer Wortgrenze beginnen.

2.3.16 CNOP Offset, Justierung

Diese Direktive justiert den Programmzähler nach dem Offsetwert und der Justierungsart. Eine Justierung von 2 heißt Wortgrenze, eine von 4 Langwort-

grenze usw. Die Justierung ist relativ zum Programmstart, der immer auf eine Langwortgrenze gesetzt wird. Zum Beispiel:

```
cnop    1,4
```

justiert den Programmzähler auf die nächste Langwort-Grenze plus 1 Byte. Beide Parameter müssen numerische, absolute Konstanten sein, Labels sind nicht erlaubt. Einige Teile des Amiga-Betriebssystems erfordern Langwort-Justierung, was mit

```
cnop    0,4
```

erreicht werden kann. Bei Programmstart wird immer langwortjustiert.

2.3.17 ORG Ausdruck

Das läßt den Assembler positionsabhängigen Code erzeugen und setzt den Programmzähler-Start auf den Wert von »Ausdruck«. Diese Option sollte mit Vorsicht verwendet werden, da das erzeugte Binär-File nicht korrekt laufen wird, weil keine Relokatiertabelle erzeugt wurde.
Die Direktive wurde aufgenommen, damit Code erzeugt werden kann, der im ROM oder auf anderen 68000-Rechnern läuft. Das erzeugte Binär-File beginnt mit dem Standard-Amiga-Header, es fehlt aber die Relokatier-Information. ORG kann nicht eingesetzt werden, wenn der Assembler linkbaren Code erzeugt.

2.3.18 <Label< DC.B Ausdruck<,Ausdruck ... u.s.w.>
<Label< DC.W Ausdruck<,Ausdruck ... u.s.w.>
<Label< DC.L Ausdruck<,Ausdruck ... u.s.w.>

Diese Direktiven definieren Konstanten im Speicher. Es sind ein oder mehrere Operanden durch Kommas getrennt möglich.
Die Konstanten werden auf Wortgrenzen im Falle von DC.W und DC.L justiert. Maximal 128 Bytes können mit einer Direktive erzeugt werden.
DC.B behandelt Strings etwas anders als bei normalen Ausdrücken. Während die schon geschilderten Regeln hinsichtlich Anführungszeichen und Hochkommas auch hier zutreffen, werden jedoch keine führenden Nullen eingefügt, und ein String kann bis zu 126 Zeichen lang sein.
Beachten Sie, daß Leerstellen in DC-Direktiven als Delimiter vor dem Komma bewertet werden. Zum Beispiel die Zeile

```
dc.b        1,2,3 ,4
```

wird nur 3 Bytes erzeugen, das vierte Byte wird als Kommentar gesehen.

2.3.19 <Label> DS.B Ausdruck
<Label> DS.W Ausdruck
<Label> DS.L Ausdruck

Diese Direktive reserviert Speicher und initialisiert dessen Inhalt mit Nullen. Wenn ein Label angegeben ist, wird dieses auf den Anfang des Bereichs gesetzt, was im Falle von DS.W und DS.L eine Wortgrenze sein wird.
Zum Beispiel reservieren diese Zeilen alle 8 Bytes auf unterschiedliche Weisen:

```
ds.b  8
ds.w  4
ds.l  2
```

2.3.20 FAIL

Diese Direktive erzeugt die Fehlermeldung »`user error`« im Pass 2. Sie kann eingesetzt werden, um zu warnen, wenn zum Beispiel die falsche Zahl von Parametern an einen Makro übergeben wurde.
Anmerkung des Übersetzers: Der Assembler merkt das natürlich auch so, die Stelle ist dann nur etwas schwieriger zu finden.

Listing-Kontrolle

Die folgenden Direktiven kontrollieren die Form des Listings.

2.3.21 LIST

Diese Direktive gibt das Listing im Pass 2 aus, und zwar auf das Gerät/File, das vorgewählt wurde, oder auf den Bildschirm, wenn vorher keine Angaben gemacht wurden. Alle folgenden Zeilen werden gelistet, bis eine END-Direktive oder das Textende erreicht ist oder eine NOLIST-Direktive. Gibt es vorher keine NOLIST-Direktive, wird auch die Symboltabelle ausgegeben, es sei denn, dies wurde via OPT-S ausgeschaltet.

2.3.22 NOLIST

NOLIST schaltet das Listing aus. Tritt danach keine LIST-Direktive mehr auf, wird auch keine Symboltabelle ausgegeben, es sei denn, daß dies über OPT-S vorgewählt wurde.

2.3.23 PLEN Ausdruck

PLEN setzt die Seitenlänge in Zeilen auf den Wert von »Ausdruck«. Voreingestellt ist 60, der neue Wert muß zwischen 12 und 255 liegen.

2.3.24 LLEN Ausdruck

LLEN setzt die Zeilenlänge in Zeichen auf den Wert von »Ausdruck«. Voreingestellt ist 132, der neue Wert muß zwischen 38 und 255 liegen.

2.3.25 TTL string

TTL druckt den Text »string«, der in Hochkommas eingeschlossen sein kann (bei Leerstellen), als Titel zu Beginn jeder Seite.

2.3.26 SPC Ausdruck

SPC erzeugt »Ausdruck«-Leerzeilen im Listing.

2.3.27 PAGE

Page bewirkt einen Seitenvorschub im Listing.

2.3.28 LISTCHAR Ausdruck, < ,Ausdruck... usw. >

Dies sendet die angegebene Zeichenfolge an das Listing-Gerät (mit Ausnahme des Bildschirms) und dient zum Beispiel zur Druckerumschaltung auf Schmalschrift. Wird zum Beispiel auf das Gerät PRT: ausgegeben, wird mit

```
listchar 27,'[','4','w'
```

jeder Drucker (der in Preferences richtig angemeldet ist) in den Schmalschrift-Modus gesetzt. Wenn Sie PAR: oder SER: benutzen, müssen Sie die druckerspezifischen Codes senden. Zum Beispiel ergibt 15 den Schmalschrift-Modus auf Epson-Druckern und Kompatiblen.

Label-Direktiven

Die folgenden Direktiven sind zusammen mit Labels anzuwenden.

2.3.29 Label EQU Ausdruck

Equate heißt soviel wie Gleichsetzung, in einigen Assemblern wird dafür tatsächlich auch das Gleichheitszeichen (=) eingesetzt.
Die EQU-Direktive setzt den Wert und den Typ von »Label« gleich dem Ergebnis von »Ausdruck«. Dieser darf keine Vorwärts-Referenzen und keine externen Labels enthalten. Bei einem Fehler in »Ausdruck« wird kein Wert zugewiesen. Das Label muß vorhanden sein.

2.3.30 Label EQUR register

Die EQUR-Direktive erlaubt es, ein Daten- oder Adreßregister mit einem vom Anwender vergebenen Namen anzusprechen, der als »Label« anzugeben ist. Dies nennt man Register-Equate. Ein Register-Equate muß vor dem ersten Gebrauch definiert werden.

2.3.31 Label SET Ausdruck

SET ist EQU ähnlich, allerdings mit dem Unterschied, daß mit SET (immer neuen SETs) einem Label andere Werte zugewiesen werden können. Vorwärts-Referenzen sind nicht erlaubt. SET ist besonders nützlich für Zähler in Makros, zum Beispiel:

```
zcount     SET   zcount+1
```

wobei unterstellt wird, daß »zcount« am Anfang des Quelltextes auf null gesetzt wurde. Zu Beginn von Pass 2 werden alle SET-Labels wieder als undefiniert initialisiert, so daß ihre Werte in beiden Durchgängen (Pass 1 und Pass 2) dieselben sind.

2.3.32 RS RSRESET RSSET

Diese Direktiven sollten Sie im Zusammenhang betrachten. Beginnen wir jedoch mit der Basis, nämlich RS.

Label	RS.B	Ausdruck
Label	RS.W	Ausdruck
Label	RS.L	Ausdruck

Die RS-Direktive erlaubt Ihnen, eine Liste von Konstanten-Definitionen zu erstellen, was sehr nützlich bei Datenstrukturen und globalen Variablen ist. Am besten zeigt man die Funktion an einem Beispiel. Angenommen, Sie greifen auf all Ihre Variablen als Offset von A6 zu (wie das in *GenAmiga* und *MonAmiga* geschieht), so können Sie diese so definieren:

```
onstate   rs.b 1
start     rs.l 1
end       rs.l 1
```

Sie können dann so zugreifen (nachdem A6 geladen wurde):

```
move.b    onstate(a6),d1
move.l    start(a6),d0
cmp.l     end(a6),d0
```

Die Direktive hat ihren eigenen internen Zähler, der bei jedem Assemblerlauf (und nach RSRESET, siehe unten) auf null gesetzt wird. Jedesmal, wenn der Assembler diese Direktive bearbeitet, weist er dem Label den aktuellen Wert dieses Zählers zu (mit Wortjustierung bei .W und .L) und inkrementiert dann den Zähler entsprechend dem Typ (.B, .W, .L) und der Größe von »Ausdruck«. Im obigen Beispiel wird »onstate« den Wert 0 haben, »start« den Wert 2 (wurde auf Wortgrenze justiert) und »end« den Wert 6.
Als weiteres Beispiel lassen Sie uns annehmen, Sie haben eine Datenstruktur, die aus einem Langwort, einem Byte und noch einem Langwort besteht, und zwar in dieser Reihenfolge. Um Ihr Programm möglichst gut lesbar und leicht änderbar zu gestalten, könnten Sie Zeilen wie diese einsetzen:

```
          rsreset
d_next    rs.l    1
d_flag    rs.b    1
d_where   rs.l    1
```

Dann können Sie so darauf zugreifen:

```
move.l  d_next(a0),a1
move.l  d_where(a0),a2
tst.b   d_flag(a0)
```

Beachten Sie bitte, daß nur ein Fehler in Ausdrücken von RS-Direktiven, wie undefiniertes Label, eine lange Folge von »label defined twice«-Meldungen erzeugen kann, weil jede folgende RS-Direktive bei jedem Durchlauf verschiedene Werte annimmt.

RSRESET

Die RSRESET-Direktive setzt den internen Zähler der RS-Direktive zurück (Reset auf 0). Sie können den aktuellen Wert dieses Zählers in Zeilen wie dieser verwenden:

rs__count rs.b 0

RSSET Ausdruck

RSSET erlaubt es, den RS-Zähler explizit auf den Wert von »Ausdruck« zu setzen. Die Direktive wird häufig in den Include-Files, die mit *Devpac* geliefert werden, eingesetzt.

Bedingtes Assemblieren

2.3.33 Allgemeines

Bedingtes Assemblieren erlaubt es, ein Programm zu schreiben, das viele interne Optionen hat, zwischen denen während des Assemblierens ausgewählt wird. Das kann aus verschiedenen Gründen sehr nützlich sein:

- Ein Programm soll auf zwei verschiedenen Maschinen laufen.
- Debug-Code soll nur für die Testversion erzeugt werden.
- Zwei leicht unterschiedliche Versionen eines Programms werden gebraucht.

Wir selbst haben all diese Features während der Entwicklung des *Devpac* gebraucht. Der Assembler läuft auf zwei Maschinen, und nur in der hausinternen Version von *GenAmiga* gibt es Debug-Code.

Es gibt viele Direktiven, die zum bedingten Assemblieren eingesetzt werden können. Zu Beginn eines »bedingten« Blocks muß eine der vielen IF-Direktiven stehen, an seinem Ende ein ENDC. Die »bedingten« Blöcke können in bis zu 32 Ebenen geschachtelt werden.
Labels sollten nicht in den IF- oder ENDC-Zeilen stehen, weil diese Direktiven dann vom Assembler ignoriert werden.

2.3.34 IFEQ Ausdruck
2.3.35 IFNE Ausdruck
2.3.36 IFGT Ausdruck
2.3.37 IFGE Ausdruck
2.3.38 IFLT Ausdruck
2.3.39 IFLE Ausdruck

Diese Direktiven entwickeln den Ausdruck, vergleichen seinen Wert mit null und schalten in Abhängigkeit vom Ergebnis das bedingte Assemblieren an oder aus. Die Bedingungen selbst entsprechen den Bedingungs-Codes des 68000. Wenn zum Beispiel das Label DEBUG den Wert 1 hat, wirken die folgenden Anweisungen

```
          IFEQ  DEBUG
logon     dc.b  'Enter a command: ',0
          ENDC
          IFNE  DEBUG
          opt    d+
logon     dc.b  'Yo man, gimme: ',0
          ENDC
```

so: Die erste Bedingung schaltet das Assemblieren aus, weil 1 nicht gleich (not EQ) 0 ist, wohingegen die zweite Bedingung das Assemblieren einschaltet, weil 1 nicht gleich (NE) 0 ist. Beachten Sie, daß IFNE dem IF in Assemblern entspricht, die nur eine Testbedingung kennen.
Die Ausdrücke müssen zu korrekten Werten entwickelbar sein. Jeder Fehler hier gilt als fatal und führt zum Abbruch des Assemblierens.

2.3.40 IFD Label
2.3.41 IFND Label

Diese beiden Direktiven erlauben die Kontrolle in Abhängigkeit davon, ob ein Label definiert ist oder nicht. Mit IFD wird das Assemblieren eingeschaltet, wenn das Label definiert ist, wohingegen mit IFND eingeschaltet wird, wenn das Label nicht definiert ist. Diese Direktiven sollten vorsichtig verwendet werden. Andernfalls kann unterschiedlicher Code in Pass 1 und Pass 2 erzeugt werden, was zu Phasenfehlern führt.

2.3.42 IFC 'string1', 'string2'

Diese beiden Direktiven vergleichen zwei Strings, die beide in Hochkommas stehen müssen. Sind beide gleich, wird das Assemblieren ein-, andernfalls wird es ausgeschaltet. Der Vergleich ist case-sensitiv (Groß-Klein-Buchstaben werden unterschieden).

2.3.43 IFNC 'string1', 'string2'

Diese Direktive verhält sich ähnlich wie die vorhergehende, nur daß das Assemblieren eingeschaltet wird, wenn die Strings nicht gleich sind. Dies sieht dem ersten Anschein nach etwas nutzlos aus, aber wenn einer oder beide Parameter Makro-Parameter sind, kann es sehr vorteilhaft sein.

2.3.44 ENDC

Diese Direktive beendet die aktuelle Ebene des bedingten Assemblierens. Wenn es mehr IF als ENDC gibt, wird eine Fehlermeldung am Ende des Assemblerlaufs ausgegeben.

Makro-Operationen

GenAmiga unterstützt voll die Makros im Motorola-Format, was zusammen mit dem bedingten Assemblieren (oder auch allein) erlaubt, die Assembler-Programmierung wesentlich zu vereinfachen und die Lesbarkeit des Programms zu verbessern.

Makro-Definitionen werden im Makro-Puffer gespeichert, welcher jeweils der freie Bereich am Ende des Textes ist, den Sie editieren. Die Anzahl freier Bytes – 4 in der Statuszeile – ist der zur Verfügung stehende Raum.

2.3.45 Label MACRO

Dies startet eine Makro-Operation und veranlaßt *GenAmiga*, alle folgenden Zeilen in den Makro-Puffer zu kopieren, bis ein ENDM auftritt. Makros können verschachtelt werden.

2.3.46 ENDM

Dies beendet (nach einer MACRO-Direktive) das Abspeichern einer Makro-Definition. Ein Label ist in dieser Zeile nicht erlaubt.

2.3.47 MEXIT

Dies beendet eine Makroentwicklung vorzeitig und ist mit dem später noch gezeigten INC-Makro bestens erläutert.

2.3.48 NARG

NARG ist keine Direktive, sondern ein reserviertes Symbol (eine Assembler-Variable). Ihr Wert ist die Anzahl der Parameter, die an das aktuelle Makro übergeben wurden oder null, wenn der Assembler nicht innerhalb eines Makros ist. Wenn *GenAmiga* im case-sensitiven Modus ist, dann ist die Schreibweise wichtig und NARG muß komplett in Großbuchstaben geschrieben werden.

2.3.49 Makro-Parameter

Wenn ein Makro mittels der MACRO-Direktive definiert wurde, kann es aufgerufen werden, indem man seinen Namen, gefolgt von bis zu 9 Parametern, als Direktive verwendet. Im Makro selbst stehen die Parameter als Backslash-Zeichen (\), gefolgt von einer Ziffer von 1 bis 9. Diese werden durch die jeweiligen Aufrufparameter ersetzt (oder auch nicht, wenn sie fehlen). Der spezielle Parameter \0 bewirkt, daß der Typ (B, W, L) beibehalten wird, der dem Makro beim Aufruf eventuell angehängt wurde (wird W, wenn der Typ fehlt).
Der Parameter \@ dient dazu, unterschiedliche Labels bei jedem Makroaufruf zu erzeugen. \@ wird in der Makroentwicklung durch _nnn ersetzt, wobei nnn eine Zahl ist, die bei jedem Aufruf um 1 erhöht wird.

Wenn ein Makro-Parameter Leerstellen oder Kommas enthält, sollte er in spitze Klammern (< >) gesetzt werden.
Per Voreinstellung wird im Assemblerlisting nur der Makro-Aufruf und nicht der produzierte Code gezeigt. Jedoch kann mit der schon beschriebenen OPT M-Direktive die Makro-Erweiterung mit Listing an- und abgeschaltet werden.
Makro-Aufrufe können in bis zu 8 Ebenen geschachtelt werden und rekursiv sein, wenn erforderlich.

2.3.50 Makro-Beispiele

Beispiel 1: Aufruf des BDOS

Im allgemeinen werden Amiga-Library-Routinen so aufgerufen:

Register A6 sichern
A6 mit dem Library-Zeiger laden
ein JSR mit Offset(A6) ausführen
A6 zurückspeichern

Ein Makro, das die o.g. Folge erledigt, könnte lauten

```
call_lib  MACRO
          move.l    a6,-(sp)
          move.l    \2,a6              Hole Lib-Zeiger
          jsr       _LVO\1(a6)         Lib-Routine aufrufen
          move.l    (sp)+,a6           A6 zurück
          ENDM
```

Die Direktiven sind nur in Großschrift, um sie hervorzuheben; diese Schreibweise ist nicht zwingend erforderlich. Wenn Sie dieses Makro nutzen wollen, um die DOS-Funktion Output aufzurufen, gilt:

```
call_lib  Output,_DOSBase
```

Wenn das Makro expandiert wird, wird \1 durch Output und \2 durch _DOSBase ersetzt. \0 -falls im Makro benutzt- würde W sein, da kein Typ im Aufruf angegeben wurde. Der o.g. Aufruf wird assembliert als

```
move.l    a6,-(sp)
move.l    \2,a6                        Hole Lib-Zeiger
jsr       _LVO\1(a6)                   Lib-Routine aufrufen
move.l    (sp)+,a6                     A6 zurück
```

Beispiel 2: Ein INC-Befehl

Der 68000 hat keinen INC-Befehl wie andere Prozessoren, aber derselbe Effekt kann mit einer »ADDQ #1«-Anweisung erreicht werden. Folgendes Makro kann dazu benutzt werden:

```
inc  MACRO
     IFC  '','\1'
     fail                                 Fehlender Parameter
     MEXIT
     ENDC
     addq.\0   #1,\1
     ENDM
```

Ein Beispiel-Aufruf wäre

```
inc.l     a0
```

dies wird expandiert zu

```
addq.l     #1,a0
```

Das Makro beginnt mit dem Vergleich des ersten Parameters gegen einen leeren String, was über FAIL eine Fehlermeldung verursacht, wenn Gleichheit besteht. Die MEXIT-Direktive wird eingesetzt, um das Makro zu verlassen, ohne den Rest zu expandieren. Wenn kein leerer Parameter existiert, wird die ADDQ-Anweisung ausgeführt, wobei der Typ des \0-Parameters eingesetzt wird.

Beispiel 3: Ein Makro zur Lösung von n-Fakultät

Obwohl man praktisch das Problem nicht so löst, definiert dieses Makro ein Label als Fakultät einer Zahl. Dies zeigt aber sehr schön, wie Rekursionen in Makros funktionieren können. Bevor das Makro gezeigt wird, ist es nützlich, sich anzusehen, wie die Lösung in einer Hochsprache wie Pascal aussehen würde:

```
function factor(n:integer):integer;
begin
     if n>0 then
               factor:=n*(factor-1)
     else
               factor:=1
end;
```

Die Makro-Definition nutzt die SET-Direktive für Multiplikationen wie n*(n-l)*(n-2) u.s.w. auf diese Art:

```
factor  MACRO
        IFND    \1
\1      set     1               setze 1 für ersten Aufruf
        ENDC

        IFGT    \2
        factor  \1,\2-1         nächste Ebenen abarbeiten
\1      set     \1*(\2)         n=n*(factor-l)
        ENDC

        ENDM

        factor  test,3          * Beispielaufruf
```

Das Ergebnis der Sache ist, daß die Variable Test auf 3! (Fakultät von 3) gesetzt wird. Der Grund dafür, daß beim zweiten SET (\2) anstatt \2 geschrieben wurde, ist, daß dieser Parameter normalerweise nicht ein simpler Ausdruck ist, sondern eine Liste von Zahlen getrennt durch Minuszeichen. Das könnte also so assembliert werden:

```
test    set   test*5-1-1-1
```

(d.h. test*5-3) anstatt des richtigen

```
        set   test*(5-1-1-1)
```

(d.h. test*2)

Beispiel 4: Bedingtes Return

Der 68000-CPU fehlt das bedingte Return, wie man es bei anderen Prozessoren findet. Aber hier kann jedoch dafür ein Makro definiert werden, der den \@-Parameter benutzt. Zum Beispiel könnte ein Makro für »return if EQ« so aussehen:

```
rtseq   MACRO
        bne.s   x\@
        rts
x\@
        ENDM
```

Der \@-Parameter wurde genommen, um unterschiedliche Labels bei jedem Aufruf zu generieren. So werden in diesem Fall Labels wie x__002 oder x__017 entstehen.

Linker-Direktiven

Wenn die Direktive OPT L+ genutzt oder die Voreinstellung mit *GenInst* geändert wurde, produziert der Assembler linkbaren anstatt ausführbaren Code. In diesem Modus sind die folgenden Direktiven zusätzlich verfügbar:

2.3.51 IDNT string

Dies setzt den Namen »string« als Modulname ein. Er kann bis 32 Zeichen lang sein, sollte keine Tab- und Leerzeichen enthalten und nicht in Anführungszeichen geschrieben werden. Nur eine IDNT-Direktive pro Modul ist zulässig.

2.3.52 SECTION string,type

Damit entsteht eine »Section« mit dem Namen »string« vom Typ »type«. Section und string können bis zu 32 Zeichen lang sein und sollten keine Tab- und Leerzeichen enthalten oder in Anführungszeichen geschrieben werden. In der aktuellen Implementation ist nur eine SECTION-Direktive pro Programm zulässig. SECTION kann bei ausführbaren Programmen zu Typvorgabe eingesetzt werden (Ausnahme BSS). In diesem Fall wird »string« ignoriert. Der Typ kann einer von den folgenden sein (in Groß- oder Kleinschrift):

CODE	Code-Section, public memory
CODE__F	Code-Section, fast memory
CODE__C	Code-Section, chip memory
DATA	Data-Section, public memory
DATA__F	Data -Section, fast memory
DATA__C	Data-Section, chip memory
BSS	BSS-Section, public memory
BSS__F	BSS-Section, fast memory
BSS__C	BSS-Section, chip memory

Code-Sections enthalten den Programm-Code, Data-Sections initialisierte Daten (Konstanten) und das BSS nicht initialisierte Daten. BSS-Sections haben den Vorteil, daß Sie keinen Platz auf der Disk belegen, nur die Größe wird gespeichert. Wenn Sie eine BSS-Section definieren, können Sie nur die DS-Direktive benutzen, jeder andere Befehl führt zu der Fehlermeldung »binary file I/O error«.

Wichtig!

Wenn Sie den Typ Chip- oder Fast-Memory einsetzen, müssen Sie Version 2.3, oder eine höhere des Linkers ALINK einsetzen, niedrigere Versionen stürzen ab.

2.3.53 XDEF label < ,label,label usw. >

Dies definiert Labels als »public«, d.h., als sichtbar (wie vorhanden) für andere Files, die zu diesem File gelinkt werden. Das Label muß innerhalb des Programmtextes definiert sein und kann relativ oder absolut (d.h. eine Konstante) sein. Ein Label kann mehrmals als XDEF deklariert werden, ohne daß dadurch eine Fehlermeldung entsteht.

2.3.54 XREF label < ,label,label usw. >

Dies definiert Labels als externe Referenzen, d.h., sie werden erst im Linkerlauf importiert und sind im Assemblerlauf unbekannt. Mit XREF werden relative Labels, d.h. Programmadressen, definiert, wohingegen mit XREF.L absolute Labels, d.h. Konstanten, definiert werden.
Ein externes Label kann in einem Befehl erlaubt eingesetzt werden, in der aktuellen Version aber nicht in einem Ausdruck. Wenn TEST also als XREF definiert wurde, dann ist die Zeile

```
move.l    TEST,d0
```

aber

```
move.l    TEST+4,d0
```

nicht und wird einen Fehler erzeugen.

Zusammenfassung der Direktiven

2.3.55 Assembler-Kontrolle

END	Ende Quelltext
INCLUDE	Liest Quelltext von Disk
OPT	Optionen folgen
EVEN	PC auf gerade Adresse
CNOP	Justiert PC
ORG	Startadresse für absoluten Code
DC	Definiere Konstante(n)
DS	Definiere Speicher
FAIL	Erzwinge Fehlermeldung

2.3.56 Listing-Kontrolle

LIST	Listing an
NOLIST	Listing aus
LLEN	Seitenlänge setzen
PLEN	Zeilenbreite setzen
TTL	Titel setzen
SPC	Leerzeilen ausgeben
PAGE	Seitenvorschub
LISTCHAR	Kontrollzeichen ausgeben.

2.3.57 Labels

EQU	Wert Label setzen
EQUR	Register-Equate
SET	Temporären Wert eines Labels setzen
RS	Reserviere Speicher
RSRESET	Reset RS-Zähler
RSSET	RS-Zähler setzen

2.3.58 Bedingtes Assemblieren

assembliere wenn

IFEQ	null
IFNE	nicht null
IFD	wenn Label definiert
IFND	wenn Label nicht definiert
IFGT	wenn größer als
IFGE	wenn größer oder gleich
IFLT	wenn kleiner als
IFLE	wenn kleiner oder gleich
IFC	wenn Strings gleich
IFNC	wenn Strings nicht gleich
ENDC	Ende bedingtes Assemblieren

2.3.59 Makros

MACRO	Start der Definition
ENDM	Ende der Definition
MEXT	Erzwingt Ende Makroexpansion

2.3.60 Linker-Direktiven

IDNT	Setze Modul-Name
SECTION	Sections-Name
XDEF	Definiert interne Labels als public
XREF	definiert externe, zu importierende Labels

2.3.61 Befehlssatz

Der 68000-Befehlssatz wird unterstützt, und einige Abkürzungen werden akzeptiert und automatisch umgesetzt. So wird die Zeile

```
cmp.w     ending,a2
```

assembliert als

```
cmpa.w    ending,a2
```

2.3.62 Wortjustierung

Alle Befehle mit Ausnahme von DC.B und DS.B werden immer auf Wortgrenzen assembliert. Benötigen Sie ein DC.B explizit auf einer Wortgrenze, sollten

Sie vorher die EVEN-Direktive schreiben. Obwohl Befehle wortjustiert sind, können Labels, denen nichts folgt, auf einer ungeraden Adresse liegen. Dies zeigt am besten ein Beispiel:

```
      nop                     ist immer wortjustiert
      dc.b   'odd'
start
      tst.l  (a0)+
      bne.s  start
```

Der o.a. Code wird nicht das gewünschte Ergebnis bringen, weil »start« einen ungeraden Wert hat. Damit solche Befehle gefunden werden können, produziert der Assembler Warnmeldungen, wenn er eine ungerade Adresse als Operand von BSR- oder BRA-Befehlen feststellt. Beachten Sie bitte, daß diese Checks nicht bei anderen Befehlen stattfinden, weshalb Sie vor solche Labels eine EVEN-Direktive setzen sollten.

Kapitel

3

MonAmiga,
der
symbolische
Debugger

3 MonAmiga, der symbolische Debugger

3.1 Einführung

Assemblerprogramme sind besonders empfindlich, weil schon der kleinste Fehler zum »Absturz« des Systems führen kann.
Um Ihnen zu helfen, diese Bugs zu finden und zu korrigieren, enthält das *Devpac*-Amiga-Paket *MonAmiga*. *MonAmiga* ist ein symbolischer Debugger und Disassembler, mit dem Sie Programme im Speicher untersuchen, Programme und einzelne Befehle ausführen und durch Programmfehler ausgelöste Exceptions des Prozessors abfangen. Weil *MonAmiga* ein symbolischer Debugger ist, können Sie sich Ihr komplettes Programm mit allen Labels ansehen. Dadurch wird das Debuggen erheblich einfacher, da man nicht mit sechsstelligen Hex-Zahlen kämpfen muß.
Obwohl *MonAmiga* ein sogenannter »Low Level«-Debugger ist, der zum Beispiel die 68000-Befehle oder -Bytes im Speicher anzeigt, kann er auch auf Programme angewandt werden, die von einer beliebigen Hochsprache compiliert wurden (und Maschinen-Code erzeugt haben). Wenn der Compiler die Option bietet, Symbole in das Binär-File zu schreiben, dann werden Sie auch Ihre Prozedur- und Funktionsnamen im Code sehen, natürlich nicht Original-Quelltext. Wir selbst benutzten *MonAmiga* zum Debuggen von *LinkAmiga*, der in C geschrieben wurde. *MonAmiga* und *GenAmiga* selbst sind komplett in Assembler geschrieben worden.
Weil *MonAmiga* seinen eigenen Bildschirmspeicher benutzt, wird das Display Ihres Programms nicht zerstört, wenn Sie es in Einzelschritten abarbeiten oder Breakpoints setzen. Das ist besonders bei graphisch orientierten Programmen wie Intuition-Anwendungen oder Spielen nützlich.

3.2 Exceptions des 68000

MonAmiga benutzt die 68000-Exceptions, um ein »Weglaufen« (einen Absturz) des zu testenden Programms abzufangen sowie für die Einzelschritt-Abarbeitung (Trace-Bit). Deshalb erscheint es sinnvoll, nun zu erklären, was beim Amiga passiert, wenn Exceptions auftreten.

Leider wurde von Commodore etwas Konfusion in die Sache gebracht, indem die Exceptions als Traps bezeichnet wurden, was aber in C und Assembler etwas völlig anderes ist.
Es gibt viele Arten von möglichen Exceptions. Einige davon sind (vom Programmierer) gewollt, andere nicht (Programmier-Fehler). Im Fall einer Exception geht der Prozessor in den Supervisor-Modus, rettet einiges auf dem Stack (SSP) und springt zur Exception-Routine. Wenn *MonAmiga* aktiv ist, richtet er einige Exception-Vektoren auf eigene Routinen, so daß er die Kontrolle übernehmen kann. Die verschiedenen Exceptions, ihr normales Ergebnis und das, was passiert, wenn *MonAmiga* aktiv ist, sind in der folgenden Tabelle dargestellt.

Exc.-Nr.	**Bezeichnung**	**Normaler Effekt**	**Effekt unter *MonAmiga***
2	Bus-Fehler	Guru	Abgefangen
3	Adreß-Fehler	Guru	Abgefangen
4	Illegaler Befehl	Guru	Abgefangen
5	0-Division	Guru	Abgefangen
6	CHK-Befehl	Guru	Abgefangen
7	TRAPV	Guru	Abgefangen
8	Privilegverletzung	Guru	Abgefangen
9	Trace	Guru	Einzelschritt
10	Line-A-Emulator	Guru	Abgefangen
11	Line-F-Emulator	Guru	Abgefangen
32–47	Trap #0-15	Guru	Abgefangen

Die genauen Gründe für die oben genannten Exceptions (und wie man sie am besten behandelt) sind detailliert am Ende dieses Abschnitts beschrieben, kurz sei jedoch zusammengefaßt:

Exceptions 2 bis 8 werden durch einen Programmierfehler verursacht und werden von *MonAmiga* abgefangen.

Exception 9 kann auf Umwegen durch einen Programmierfehler ausgelöst werden. Sie wird von *MonAmiga* für die Einzelschritt-Bearbeitung benutzt.
Der Rest (d.h. die Trap-Befehle) sind auf *MonAmiga* umgelenkt, können aber nachträglich umdefiniert werden, wenn es das Programm erfordert.

»Guru« in der oben genannten Tabelle heißt, daß der Amiga versuchen wird, einen System-Alert-Requester auf den Schirm zu zeichnen, der die Exception-Nummer und den PC-Wert zeigt. Gelegentlich wird bei schweren Fehlern der ganze Schirm mit buntem Müll gefüllt, was zwar sehr beeindruckend aussieht, aber nicht sehr nützlich ist.

3.3 Starten von *MonAmiga*

MonAmiga wird gestartet durch die Eingabe des Kommados

```
monam [Return]
```

Dem kann wahlweise ein Programm-Name oder eine Kommandozeile (für das Programm) folgen. Zum Beispiel

```
monam c:genam examples/demo.s
```

startet *MonAmiga*, lädt *GenAmiga* und übergibt letzterem einen File-Namen. Nachdem *MonAmiga* geladen wurde, zeigt der Bildschirm vier Fenster, in denen *MonAmiga* alle Informationen darstellt. Jedes Fenster hat einen speziellen Zweck, doch vorerst ist das unterste Fenster das wichtigste. Hier erscheinen alle Tastatureingaben, während die anderen Fenster diverse Meldungen ausgeben.
Direkt nach dem Laden erscheint die Aufforderung »Enter program name or Return:«. An dieser Stelle müssen Sie entscheiden, ob Sie ein Programm debuggen wollen oder sich nur den Speicher ansehen wollen.

3.4 Programm debuggen

Wenn Sie ein bestimmtes Programm debuggen wollen, sollten Sie den File-Namen einschließlich Laufwerksangabe und Extender eingeben, und *MonAmiga* wird versuchen, den File zu laden. Schlägt das fehl, erscheint die Meldung

```
AmigaDOS error xx
```

und Sie können einen anderen File-Namen eingeben oder nur [Return], wenn Sie Ihre Meinung geändert haben und kein Programm debuggen wollen. Angenommen, der File-Name ist gültig, wird ihn *MonAmiga* laden. Existiert eine Symboltabelle, erscheint die Meldung »`Symbols loaded`« im obersten Fenster. Nach dem erfolgreichen Laden erscheint der Prompt »Enter command line:« und Sie können eine Kommandozeile eingeben, die wie üblich (d.h. in die Base Page) übergeben wird. Wenn Sie keine Kommandozeile wünschen, geben Sie nur [Return] ein. Nun kommt die Meldung

```
Exception: Breakpoint
```

und es erscheint das Haupt-Display. Der Grund für den Breakpoint ist, daß *MonAmiga* einen Breakpoint auf den ersten Befehl des Programms setzt und dann dahin springt.

3.5 *MonAmiga* und Multitasking

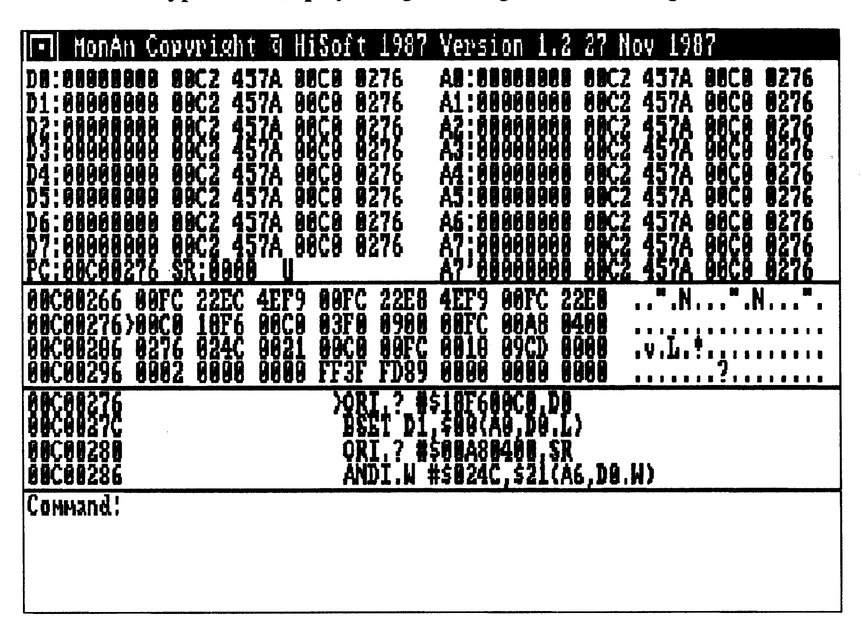
Der Amiga ist Multitasking-fähig, und das hat für *MonAmiga* einige Einschränkungen zur Folge. Wurde ein Programm (eine Task) geladen, wird er in einen Wartezustand versetzt, das heißt in diesem Fall, er wartet auf ein *MonAmiga*-Kommando, das ihn weiterlaufen läßt. Der andere Zustand ist, daß die Task gleichzeitig mit *MonAmiga* läuft. Einige Kommandos funktionieren nur jeweils in einem dieser beiden Zustände. Zum Beispiel können Sie eine Task in Einzelschritten nur im Wartezustand abarbeiten. Läuft er, wenn Sie das Kommando (für single step) eingeben, kommt die Fehlermeldung »`Task must suspended!`«.

3.6 Speicher ansehen

Wenn Sie sich den Speicher ansehen wollen, geben Sie nur [Return] ein, und Sie sehen das Haupt-Display und den Prompt »Command:«.

Das Haupt-Display

Das Haupt-Display von *MonAmiga* zeigt Register, einen Speicherbereich und Befehle. Ein typisches Display bringt die folgende Abbildung.



3.7 Register-Anzeige

Das oberste Fenster zeigt die Daten- und Adreßregister an sowie den SSP, den Programmzähler und das Statusregister. Für jedes Register (ausgenommen SR und PC) wird der Wert angezeigt, gefolgt von 8 Bytes des Speichers. Die Daten sind wortjustiert, um sie einfacher erfassen zu können. Die Werte von PC und SR werden zusammen mit den Flags dargestellt. Zusätzlich werden die »Flags« T für Trace, S für Supervisor-Modus und U für User-Modus gezeigt, wenn sie relevant sind.

3.8 Speicher-Anzeige

Das zweite Fenster zeigt einen Speicherbereich um den sogenannten »Memory Pointer« (Speicherzeiger) herum an. Es zeigt 4 Zeilen von 16 Bytes Länge an und beginnt mit einer Zeile vor dem Memory Pointer. Der Speicher wird wortjustiert in hex dargestellt, dazu kommt rechts die ASCII-Darstellung. Das Byte, auf das der Memory Pointer zeigt, ist mit dem Zeichen > markiert. Beim Start zeigt der Memory Pointer auf dieselbe Adresse wie der PC, dasselbe nach jeder Exception.

3.9 Befehls-Anzeige

Das dritte Fenster zeigt einen Speicherauszug vom Memory Pointer an, in disassemblierter Form über vier Befehle. Wenn der Memory Pointer ungerade ist, startet der Disassembler mit dem nächsten Byte, weil Befehle nicht auf ungeraden Adressen beginnen können.

3.10 Das Kommando-Fenster

Im untersten Fenster erfolgen alle Tastatureingaben. Hier erscheinen auch diverse Prompts, die Kommandos (Eingaben) abfragen.

Kommando-Eingabe

Kommandos werden über einen Tastendruck nach dem Prompt »Command: « in Groß- oder Kleinschrift eingegeben. Kommandos, die zerstörende Wirkung haben können, erfordern zusätzlich die Ctrl-Taste, die gleichzeitig mit der Kommandotaste gedrückt werden muß. Die Kürzel wurden soweit wie möglich so gewählt, daß man sie sich leicht merken kann. Kommandos wirken sofort, [Return] ist nicht erforderlich. Ungültige Kommandos werden einfach ignoriert. Die relevanten Teile des Haupt-Displays werden aktualisiert, sofern ein sofort sichtbarer Effekt bewirkt wird.

3.11 String-Eingabe

Eine Stringeingabe kann mit der Backspace-Taste korrigiert werden. Escape bricht die Eingabe ab.

3.12 Zahlen-Eingabe

Wenn eine Zahl eingegeben werden muß, kann das auf verschiedene Weisen geschehen. Als Standard werden Hex-Zahlen angenommen. Dezimalzahlen müssen mit dem #-Zeichen beginnen. Wenn eine Symboltabelle geladen wurde, können Labels mit einem vorgestellten Punkt eingegeben werden. Beachten Sie, daß bei Labels Groß- und Kleinschrift unterschieden wird und daß nur die ersten 16 Zeichen von Bedeutung sind. Wenn Sie Labels haben, die in den ersten 16 Zeichen gleich sind, wird das zuerst von der Disk geladene benutzt, was wiederum von dem Programm abhängt, das das Binär-File erzeugt hat. Wenn *GenAmiga* den Code erzeugt hat, werden die Labels in alphabetischer Reihenfolge geladen, aber in anderen Programmen kann die Folge undefiniert sein.

3.13 Reservierte Labels

Neben den anwenderdefinierten Labels, die von der Disk geladen werden, gibt es noch reservierte Labels, die nicht case-sensitiv sind. Sie werden auch mit einem führenden Punkt eingegeben. Es gibt:

CODE	Start des Programms
SSP	Supervisor Stack Pointer
USP	User Stack Pointer
D0..D7	Datenregister
A0..A7	Adreßregister
PC	Programmzähler

Weil Ihre Symboltabelle vor dieser reservierten Tabelle durchsucht wird, wird *MonAmiga* ein von Ihnen eingesetztes, reserviertes Label, wie zum Beispiel CODE, auch verwenden. Um auf das »reservierte CODE« zugreifen zu können, geben Sie einfach »code« ein. Das wird in der Symboltabelle nicht gefunden, wohl aber in der reservierten Tabelle, weil diese nicht case-sensitiv ist.

MonAmiga-Kommandos

Die verschiedenen *MonAmiga*-Kommandos werden nun beschrieben, wobei immer eine Seite für eine Kommando-Gruppe vorgesehen ist.

3.14 Crtl-C *MonAmiga* verlassen

Damit kommen Sie direkt zum CLI zurück. Wenn die Task, die Sie debuggen, noch läuft oder im Wartezustand ist, werden Sie gewarnt. Wenn *MonAmiga* beendet wird, während eine Task läuft, stürzt der Rechner ab, wenn zusätzlich noch eine Exception auftritt. Sicherer ist, die Task zuerst mit Ctrl-Q anzuhalten.

3.15 Ctrl-Q Programm beenden

Damit kann der Abbruch der Task erzwungen werden. Dies kann gefährlich werden und sollte nur als letztes Mittel eingesetzt werden. Wenn ein Programm so beendet wird, fehlt das übliche »Clean up«, sprich: Fenster werden nicht geschlossen, der Speicher wird nicht freigegeben usw.

3.16 Ctrl-S Stoppen eines Programms

Dieses Kommando stoppt Ihre Task während der Ausführung. Das geschieht durch Setzen des Trace-Bits, weshalb Sie auch eine Trace-Exception bekommen.

Wichtiger Hinweis

Die beiden o.g. Kommandos benutzen Speicherbereiche, für die nicht garantiert ist, daß sie die gleichen für verschiedene Betriebssystem-Versionen sind. Deshalb sollten Sie zuerst mit unwichtigen Daten im Rechner die Kompatibilität zu Ihrer Version testen, bevor Sie gezwungen sind, die Befehle anzuwenden, wenn eine Task verrückt spielt.

Ändern des Speicherzeigers

3.17 M Modifizieren des Speicherzeigers

Das Kommando fragt Sie nach einer Adresse und aktualisiert dann den Schirm, um den Stand anzuzeigen. M ist zwingend erforderlich für die Eingabe von Code, das Ansehen von Speicherbereichen usw.

3.18 [Return] Inkrementieren des Speicherzeigers

Dies inkrementiert den Speicherzeiger um 1 Wort (oder um 1 Byte, wenn er ungerade war).

3.19 – (minus) Dekrementieren des Speicherzeigers

Dies dekrementiert den Speicherzeiger um 1 Wort (oder um 3 Bytes, wenn er vorher ungerade war).

3.20 T Temporäres Update des Speicherzeigers

Dies erlaubt es Ihnen, temporär den Speicherzeiger auf den Wert eines Registers zu stellen. Sie werden mit »Register:« danach gefragt. Sie können D (Datenregister), A (Adreßregister) oder P (für PC) eingeben. Für Daten- und Adreßregister muß direkt eine Ziffer von 0 bis 7 folgen. Nach einer passenden Registerwahl wird das Haupt-Display aktualisiert, um den neuen Stand anzuzeigen, und die Frage »`Update Y/N?`« erscheint. Mit Y wird der aktuelle Stand permanent, jede andere Taste stellt den Speicherzeiger wieder auf den ursprünglichen Wert.

3.21 [Tab] Speicherzeiger auf PC setzen

Tab stellt den Speicherzeiger und somit das Display wieder auf den Wert des PC. Dies ist nützlich, wenn man schnell zum Programm-Code zurück will, nachdem man sich einen Speicherbereich angesehen hat.

Speicheranzeige

3.22 X Speicher eXaminieren

X zeigt einen Speicherbereich »ab Speicherzeiger« in hex und ASCII. Es füllt eine volle Bildschirmseite mit Zeilen von 16 Bytes, gefolgt von deren ASCII-Äquivalenten. Nach einer Schirmseite wartet *MonAmiga* auf einen Tastendruck und zeigt dann die nächste Seite. Mit der Escape-Taste kommen Sie wieder zum Haupt-Display.

3.23 Ctrl-X Speicher eXaminieren mit Druckerausgabe

Dies funktioniert sinngemäß wie X, nur daß die Ausgabe zum Drucker geht, solange Sie nicht mit Escape stoppen. Nach dem Start kennt *MonAmiga* keinen »Printer«, weshalb Sie vorher Ctrl-P und dann einen gültigen Amiga-Gerätenamen wie PRT: eingegeben haben sollten. Da Amiga-DOS geräteunabhängiges I/O unterstützt, hindert Sie jedoch nichts daran, einen File-Namen einzugeben und auf die Disk zu »drucken«.

3.24 Q Schnelles (Quick) Disassemblieren

Q zeigt einen Bereich »ab Speicherzeiger« in disassemblierter Form, inklusive vorhandener Labels über eine Schirmseite. Nach Druck auf eine beliebige Taste folgt die nächste Seite. Mit Escape kommen Sie wieder zum Haupt-Display zurück.

3.25 P Disassemblieren zum Drucker

P gibt ein Disassembler-Listing auf den Schirm oder den Drucker aus. Sie werden nach der Start- und der Endadresse gefragt. Ist die Startadresse größer als die Endadresse, wird das Kommando abgebrochen. Danach folgt der Prompt »Printer Y/N?«. Bei Y erfolgt die Ausgabe auf den Drucker, sonst auf den Schirm. Die Ausgabe auf jedes Gerät kann mit Escape abgebrochen werden.

Speicher durchsuchen

3.26 G Finde (Get) Folge

Nach G folgt der Prompt »Search for B/W/L/T/I?«, was für Byte, Wort, Long, Text und Befehl (Instruction) steht.
Wenn Sie B, W oder L wählen, werden Sie nach der Zahlenfolge gefragt, die gesucht werden soll. An deren Ende geben Sie [Return] und die Suche beginnt. *MonAmiga* besteht nicht auf Wortgrenzen, weshalb zum Beispiel auch Langworte auf ungeraden Adressen gefunden werden.
Wenn Sie T wählen, werden Sie nach einem String gefragt. Die Suche ist casesensitiv.
Mit I suchen Sie einen Teil einer Mnemonik eines Befehls. So können Sie zum Beispiel, wenn Sie nach »14(A« suchen, »MOVE.L D2,$0014(A0)« finden. Die Schreibweise (groß/klein) ist unwichtig, aber Sie sollten das Format des Disassemblers beachten, der nur Hex-Zahlen benutzt, A7 anstatt SP schreibt usw.
Nach der Wahl eines Suchbefehls und seiner Parameter beginnt die Suche, die dann die Kontrolle an die N-Routine (siehe folgende) übergibt.

3.27 N Nächste Folge finden

N kann genutzt werden, um ein weiteres Auftreten des Suchbegriffs im Speicher zu suchen. Mit den Optionen B, W, L und T werden Sie die Folge wenigstens einmal finden, nämlich in dem Puffer, in dem sie *MonAmiga* abgelegt hat.

Bei diesen Optionen wird die Escape-Taste, mit der Sie die Suche stoppen können, alle 64 Bytes abgefragt. Bei der I-Option, die sehr viel langsamer ist, wird die Escape-Taste alle zwei Bytes geprüft.
Der durchsuchte Speicherbereich geht von 0 bis $200000 (2 Megabyte), dann von $F00000 bis $FFFFFF (ROM), und beginnt dann wieder bei 0.

Code ausführen

Bitte beachten Sie, daß alle Befehle auf dieser Seite verlangen, daß die Task im Zustand »suspended« (wartend) ist.

3.28 R Register setzen

R erlaubt, die angezeigten Register zu ändern. Sie werden nach dem Register gefragt, wie beim T-Kommando beschrieben. Hinzu kommt S für Statusregister. Dann werden Sie nach einem Wert für dieses Register gefragt, worauf das Register entsprechend gesetzt wird. Beachten Sie, daß der SSP nicht geändert werden kann, weil dieser sehr wichtig für die korrekte Funktion von *MonAmiga* und des Betriebssystem überhaupt ist.

3.29 Ctrl-Z Befehle im Einzelschritt

Dies führt den Befehl aus, auf den der Speicherzeiger gerade weist, wobei die angezeigten Registerwerte benutzt werden. Einzelschritt ist für jeden gültigen Befehl möglich. Die Routine nutzt die Trace-Funktion des 68000.

3.30 Ctrl-R Rückkehr zum Programm

Damit kehrt *MonAmiga* zu der Stelle zurück, die die Exception ausgelöst hatte. Das kann ein Breakpoint gewesen sein, eine Einzelschritt-Bearbeitung oder die Korrektur eines Adreßfehlers. Wenn Sie zu einer bestimmten Adresse springen

wollen, können Sie den PC setzen (siehe oben genanntes R-Kommando) und dann Ctrl-Z geben, was ein Return zu dieser Adresse bewirkt und somit die Ausführung dort fortsetzt.

3.31 Ctrl-T Interpretieren eines Befehls

Ctrl-T führt einen Unterprogramm-Aufruf aus, d.h., das Unterprogramm wird im Stück abgearbeitet. Das Kommando darf deshalb auch nur auf die Befehle BSR, JSR und Trap angewandt werden. Es ist bei RAM- und ROM-Routinen zulässig, speziell bei letzteren kann man sich damit zahllose »Ctrl-Z« ersparen.

Breakpoints

Unterbrechungspunkte

Breakpoints in *MonAmiga* ersetzen das erste Wort eines Befehls mit dem Code für »illegaler Befehl« ($4AFA), natürlich nachdem das Original in einer Tabelle gesichert wurde, aus der es später wieder zurückgeladen wird. Breakpoints können nicht auf ungerade Adressen gesetzt werden und auch nicht auf Adressen in Nur-Lesespeichern (Kickstart-RAM oder ROM). Bis zu 8 Breakpoints gleichzeitig sind möglich.

3.32 Ctrl-B Setzen/Rücksetzen von Breakpoints

Dieses Kommando setzt entweder einen Breakpoint auf die Adresse des Speicherzeigers, oder löscht ihn, falls da schon einer ist. Dabei kann eine von drei möglichen Fehlermeldungen erscheinen, nämlich »`Its odd!`« (ungerade Adresse), »`Cannot write!`« (ROM) oder »`Too many-1`« (zu viele, mehr als 8). Jede dieser Meldungen bricht das Kommando ab. Wenn ein Breakpoint im laufenden Programm erreicht wird, erscheint die Nachricht »`Exception: Breakpoint`« im Kommandofenster, gefolgt vom Prompt »Command: «.

3.33 Ctrl-K Löschen von Breakpoints

Dieses Kommando löscht alle Breakpoints, indem es jedes $4AFA durch das ursprüngliche Wort ersetzt.

3.34 Ctrl-A Einfügen von Breakpoints und Ausführung

Dieses Kommando setzt einen Breakpoint nach dem Befehl, auf den der Speicherzeiger zeigt, und führt dann den Befehl vor dem Breakpoint aus. Dies ist besonders bei DBcc-Schleifen vorteilhaft, wenn man nicht alle Iterationen in Einzelschritten abarbeiten, sondern nur das Ergebnis nach dem letzten Durchlauf sehen will. Das Kommando kann dieselben Fehlermeldungen wie Ctrl-B generieren. Zusätzlich darf es nur auf die wartende Task angewandt werden.

3.35 U go until (laufe bis)

Sie werden nach einer Adresse gefragt, und auf diese wird ein Breakpoint gesetzt. Das Programm läuft dann bis dahin oder bis zu einem vorherliegenden Breakpoint. Gestartet wird mit den angezeigten Registerwerten ab Adresse PC.

Weitere Kommandos

3.36 B BCPL-Zeiger-Konvertierung

Dieses Kommando ist nützlich für die Konvertierung von und in BCPL-Zeiger, die häufig in Amiga-DOS benutzt werden. Sie werden nach einer Zahl gefragt (Number to convert?). Danach wird die Zahl mal 4 (oder um 2 Bit nach links geschoben, was immer Sie vorziehen) und dividiert durch 4 angezeigt.

3.37 I Intelligente Kopie

Es wird ein Block von einer Speicheradresse auf eine andere kopiert. Der Vorgang ist insofern intelligent, als daß sich Quell- und Zielbereich auch überlappen können. Das Kommando fragt nach der Start- und Endadresse (First?, Last?) des Quellblocks und dann, wohin (To?) kopiert werden soll. Wenn die Startadresse größer als die Endadresse ist, wird das Kommando abgebrochen, andernfalls wird der Block kopiert, wie angegeben. Beachten Sie, daß Vorsicht angebracht ist, weil keinerlei Prüfungen stattfinden, ob die Kopie gültig ist. Kopien auf nicht existierende Speicherbereiche oder auf Systemvariable sind möglich, entsprechend dann auch der Systemabsturz.

3.38 W Speicher mit Muster füllen (Write)

Das Kommando fragt nach »First, Last, With« (Startadresse, Endadresse, womit). Der Bereich wird mit dem Byte von »womit« gefüllt. Die Warnung bezüglich fehlender Checks gilt wie beim W-Kommando auch hier.

3.39 L Labels (Symboltabelle) auslisten

Dieses Kommando listet alle geladenen Labels mit ihren Werten auf. Nach einer Schirmseite wird auf einen Tastendruck gewartet, Escape bricht das Listing ab.

3.40 Ctrl-L Labels (Symboltabelle) freigeben

Dieses Kommando kann nur auf Tasks angewandt werden, die eine Symboltabelle geladen haben. Es gibt den Speicher frei, den die Symboltabelle belegt (hat), womit beim Debuggen sehr großer Programme Speicher für das Programm freigemacht werden kann, der sonst eventuell fehlt. Man kann somit ein Programm mit der Symboltabelle laden, einen Breakpoint (ganz bequem) auf ein Label setzen, und dann die Symboltabelle entfernen und das Programm starten. Natürlich stehen keine Symbole mehr zur Verfügung, wenn der Breakpoint erreicht ist.

3.41 H Hilfsanzeige

Dieses Kommando zeigt den freien Speicher an und – wenn ein Programm geladen ist – Programmnamen und Hunk-Liste. Zu »Hunk« siehe Amiga-DOS Developers Manual. Ein Programm kann aus vielen Hunks bestehen, die im RAM gestreut abgelegt sind. Mit »H« erhalten Sie dazu eine Liste.

3.42 O Andere (Other) Zahlenbasen

Sie werden nach einer Zahl gefragt, die Sie in jeder erlaubten Notation eingeben können. Die Zahl wird dann in hex, dezimal und als Symbol (wenn relevant) angezeigt.

Speicher modifizieren

Um Daten in den Speicher einzugeben, setzt man mit dem M-Kommando den Speicherzeiger auf die gewünschte Adresse und kann dann nach dem C-Kommando Bytes, Worte oder Langworte eingeben.

3.43 C Speicher ändern (Change)

Dies ist das Kommando, mit dem Bytes, Worte oder Langworte direkt in den Speicher eingegeben werden können. Nach C erscheint die Frage »Bytes/Words/Longs?«, die Sie mit B, W oder L beantworten müssen. Sie können danach eine Liste von Wörtern eintippen, von denen jedes mit [Return] abzuschließen ist. Die Eingabe wird beendet, wenn Sie nur [Return] eingeben oder die Escape-Taste drücken.

3.44 " Direkte Stringeingabe in den Speicher

Nach dem Anführungszeichen (") können Sie direkt Text »ab Adresse Speicherzeiger« eingeben. Die Eingabe wird mit [Return] beendet, das CR-Zeichen wird jedoch nicht eingelesen. Wenn Sie den String mit einem CR oder einem Null-Byte terminieren wollen, müssen Sie dieses Zeichen mit dem C-Kommando eingeben.

3.45 Einschränkungen

MonAmiga benötigt die Exec- und Graphic-Library. Wenn Ihr Programm Speicherbereiche zerstört, auf die es nicht zugreifen darf, können wichtige Systemvariable ausfallen. Wenn dann *MonAmiga* über eine Exception angesprungen wird, wird das System abstürzen. Glücklicherweise ist diese Art Fehler sehr selten. Normalerweise treten Adreß-Fehler auf (die fängt *MonAmiga* ab), bevor Speicherbereiche zerstört werden.
Wenn ein Programm von *MonAmiga* aus gestartet wird, sieht es immer wie ein vom CLI aus aufgerufenes Programm aus, nicht wie eines, das von der Workbench gestartet wurde.
MonAmiga kann kein Programm im Einzelschritt abarbeiten, das im Supervisor-Modus läuft. Auch Breakpoints sind dann nicht möglich. Der Grund ist, daß der Exception-Handler von Exec diese Möglichkeiten nicht zuläßt und in diesem Fall einen System-Alarm (Guru) auslöst. Exec ignoriert dann den *MonAmiga*-Aufruf und arbeitet normal weiter .
Wenn Ihr Programm eine andere Task öffnet, können Sie mit *MonAmiga* diese Task nicht im Einzelschritt oder mit Breakpoints abarbeiten. *MonAmiga* kann immer nur das Programm bearbeiten, das beim Aufruf von *MonAmiga* geladen wurde.
Probieren Sie nicht, die Standard-System-Programme wie zum Beispiel DIR unter *MonAmiga* laufen zu lassen. Diese Programme basieren auf undokumentiertem Umgang mit Registern (speziell A5) und Speicherbereichen, die *MonAmiga* nicht emulieren kann.
Aufgrund eines Hardware-Features führt ein Wort- oder Langwortzugriff auf ungerade Adressen zwischen 1 und 7 zu einem totalen Systemzusammenbruch.
MonAmiga liest zur Zeit nur den ersten Debug-Hunk ein, den es in einem Programm-File findet. Files, die mit *GenAmiga* erstellt wurden, haben auch nur einen Hunk, genauso wie die von Aztek-C. Jedoch gibt es Files mit mehreren Debug-Hunks, was beim Linken oder in Lattice-C entstehen kann. In diesen Fällen müssen Sie damit rechnen, daß die eingelesenen Debug-Informationen unvollständig sind.

Aufgrund des erweiterten Stack-Rahmens bei Exceptions kann die derzeitige *MonAmiga*-Version nicht Programme unter einem 68010 oder 68020 abarbeiten. Sie kann aber noch zur Speicher-Examinierung und zum Disassemblieren eingesetzt werden.

3.46 Exception-Nachbehandlung

Wenn eine unerwartete Exception ausgelöst wurde, sollte man herausfinden können, wo und warum sie auftrat, um dann (nach geeigneten Maßnahmen) möglichst das Programm weiterlaufen lassen zu können. Dazu einige Hinweise:

Adreß-Fehler

Nach diesem Fehler zeigt der angezeigte Wert des PC auf die Adresse innerhalb des Befehls, auf der ein Zugriff auf eine ungerade Adresse versucht wurde. Das ist normalerweise nicht der Beginn des Befehls. Deshalb sollte man dann mit dem – (Minus)-Kommando soweit zurückgehen, bis der wirkliche Befehl gefunden wird. Wenn Sie diesen Befehl noch einmal ausführen wollen (nachdem Sie Register und/oder Speicheradressen geändert haben, damit der Fehler nicht nochmals auftritt), so sollten Sie den PC mit dem R-Kommando ändern und dann Ctrl-R eingeben oder den Speicherzeiger ändern und mit Ctrl-Z den Befehl im Einzelschritt ausführen.

Illegaler Befehl

Danach zeigt im Haupt-Display der Speicherzeiger auf den illegalen Befehl. Sie können diesen nun korrigieren oder den PC einen Befehl weitersetzen, bevor Sie mit Ctrl-R das Programm fortsetzen.

Privileg-Verletzung

Danach zeigt das Haupt-Display den Befehl, der im User-Modus nicht ausgeführt werden konnte (privilegierte Befehle sind nur im Supervisor-Modus zulässig). Dies bedeutet normalerweise, daß Ihr Programm einen grundsätzlichen Fehler enthält. Nun den PC auf den nächsten Befehl zu stellen und dann mit Ctrl-R fortzufahren, kann nur eine temporäre Lösung sein.

Verschiedene Fehler

Es kann vorkommen, daß Programme den falschen Speicherbereich ansprechen. Dann kann jede der o.g. Exceptions auftreten, abhängig davon, was im Speicher war, als Ihr Programm abstürzte. Wenn dies eintritt, kann der PC eine Hausnummer wie $00000000 oder $A0000000 sein. Wenn dieser Wert sinnlos ist, ist auch der Stack »außer Tritt«, und es wurde versucht, ein RTS auszuführen. Dieses landete auf einer Zufallsadresse, was dann den Fehler auslöste. Um herauszufinden, wo Sie nun wirklich herkamen, sollten Sie sich den Stack ansehen und versuchen, eine »echte« Return-Adresse zu finden. In diesem Bereich sollten Sie dann nach der Ursache für den falschen Wert des Stackpointers suchen.

3.47 Zusammenfassung der *MonAmiga*-Kommandos

B	BCPL-Pointer-Konvertierung
C	Change (ändere) Speicher
G	Get (suche) Folge
H	Hilfe
I	Intelligente Kopie
L	Labels auflisten
M	Modifiziere Speicherzeiger
N	Nächstes finden (nach G)
O	Other (andere) Zahlenbasis zeigen
P	Print Disassembler-Listing auf Drucker oder Schirm ausgeben
Q	Quick (schnelles) Disassemblieren
R	Register-Wert ändern
T	Temporär Speicherzeiger ändern
U	go Until (bis)
W	fülle Speicher mit (with)
X	eXaminiere Speicher
Ctrl-A	Breakpoint nach aktuellem Befehl, Befehl ausführen
Ctrl-B	Breakpoint setzen/löschen
Ctrl-C	*MonAmiga* verlassen
Ctrl-K	Kill (lösche) alle Breakpoints
Ctrl-L	Von Symboltabelle belegten Speicher freigeben
Ctrl-P	Printer (oder File) für Ausgabe angeben
Ctrl-Q	Quittieren des Programms (im Debug-Lauf) erzwingen
Ctrl-R	Resume (Fortsetzen) nach Exception
Ctrl-S	Stoppe weglaufendes Programm

Ctrl-T	Unterprogramm (und Trap) ausführen
Ctrl-X	Speicher auf Drucker (oder File) ausgeben
Ctrl-Z	Einzelschritt
"	ASCII-Eingabe in Speicher
[Return]	Speicherzeiger um 1 Wort inkrementieren
–	Speicherzeiger um 1 Wort dekrementieren
[Tab]	Speicherzeiger auf Wert von PC

Kapitel

4

Das Installations-Programm

4 Das Installations-Programm

Wichtig!

Zum *Devpac* gehört ein Installations-Programm, das Ihnen erlaubt, *GenAmiga* an Ihre individuellen Anforderungen anzupassen. Es modifiziert das *GenAm*-Programm selbst, weshalb es niemals mit der Original-Diskette eingesetzt werden sollte, sondern immer nur mit einer Kopie.

4.1 Anwendung des Installationsprogramms

Um das Programm zu starten, geben Sie ein:

```
geninst [Return]
```

Das Programm wird geladen und bringt dann einen File-Requester, der nach dem zu installierenden File fragt. Voreingestellt ist »C:GenAm«. Ändern Sie dies, falls notwendig, oder klicken Sie auf OK. Das Programm lädt dann *GenAmiga* und zeigt anschließend die verschiedenen änderbaren Parameter in einem Fenster auf der Bildschirmmitte.

4.2 Textgröße

Dies ist die Größe des Textpuffers des Editors, ursprünglich 60000 Bytes. Wenn Sie über eine RAM-Erweiterung verfügen, können Sie den Bereich beliebig groß wählen, jedenfalls solange ein kontinuierlich großer Speicherbereich vorhanden ist. Vergessen sie nicht, Platz für die während des Assemblierens entstehende Symboltabelle zu lassen, Minimum sind 1000 Bytes. Wenn Sie sie größer als 150000 Bytes machen, werden Sie Wartezeiten beim Einfügen neuer Zeilen zu Beginn großer Files bemerken, weil der Editor dann mehr Speicher verschieben muß. Bitte beachten Sie: *GenAmiga* muß die gewünschte Anzahl Bytes in einem Stück belegen können. So kann es vorkommen, daß zum Beispiel bei 200000 »Bytes free« 150000 Bytes nicht belegt werden können, weil der freie Speicher sich aus vielen kleinen Stücken zusammensetzt.

4.3 Tabulatoren

Voreingestellt ist ein Tabulatorabstand von 8. Sie können dies in einen Wert zwischen 2 und 16 einschließlich ändern.

4.4 XREFs

Dies ist die maximale Anzahl externer Labels. Voreingestellt ist 50. Jedes Label belegt 4 Bytes, wenn *GenAmiga* geladen ist. Sie können die Anzahl auf 0 setzen, wenn Sie keinen linkbaren Code erzeugen wollen oder Sie können ihn größer wählen, wenn Sie beabsichtigen, sehr viele externe Labels einzusetzen.

4.5 Code-Erzeugung

Es gibt zwei Arten von Code, »Executable Code« (ausführbar) und »Linkable Code«. Der erstere ist vorgewählt. Dies bestimmt den Typ, den *GenAmiga* beim Assemblieren produziert. Das kann allerdings überschrieben werden durch die Direktive OPT L im Quelltext oder in der Kommandozeile mittels –l bzw. +l.

4.6 Groß-Klein-Schrift-Unterscheidung

Die beiden nächsten Knöpfe bestimmen, ob *GenAmiga* während des Assemblierens case-sensitiv ist (Groß-Klein-Buchstaben unterscheidet) oder nicht. Sie können dies Ihren Anforderungen entsprechend einstellen. Jede Einstellung kann mit der Direktive OPT C im Quelltext oder mit +c bzw. –c in der Kommandozeile überschrieben werden.

4.7 Fenstergröße

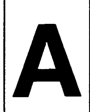
Ein Klick auf »default size« schaltet die Fenstergröße des Editors um auf »full-screen-size« (volle Schirmgröße), noch ein Klick schaltet wieder zurück. Damit können deutsche Amigas (PAL) die zusätzlichen 56 Zeilen ausnutzen. Auch im Interlace-Betrieb ist diese Einstellung vorteilhaft.

Wenn Sie auf »Quit« klicken, wird *GenAmiga* nicht geändert. Nach dem Anklicken von »Save« werden Ihre Änderungen übernommen. Wenn Sie den geänderten Stand unter einem anderen Namen abspeichern wollen, müssen Sie den Namen im Feld »Save installed version as« vorher ändern.

Anhang

A

Amiga-DOS-Fehlernummern



Anhang: Amiga-DOS-Fehlernummern

Dargestellt sind die Fehlernummern, die teilweise erscheinenden englischen Texte, deren Übersetzung, und, wenn nötig, zusätzliche Erklärungen.

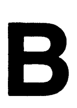
103	`insufficent free store`	Nicht genug freier Speicher
104	`task table full`	Task-Tabelle voll (mehr als 20 CLIs)
120	`argument line invalid or too long`	Argument-Zeile ungültig oder zu lang (bei CLI-Kommandos)
121	`file is not an object module`	File ist kein Objekt-Modul (kann nicht ausgeführt werden)
122	`invalid resident library during load`	ungültige residente Library während des Ladens festgestellt
202	`object in use`	Objekt in Gebrauch (z.B. File durch anderes Programm)
203	`object already exists`	Objekt existiert bereits
204	`directory not found`	Directory nicht gefunden
205	`object not found`	Objekt nicht gefunden (File nicht gefunden)
206	`invalid window`	ungültige Window-Spezifikation (CON:/RAW:-Window)
209	`packet request type unknown`	Typ unbekannt
210	`invalid stream component name`	Name ist zu lang oder enthält Ctrl-Zeichen
211	`invalid object lock`	Ungültiges Lock
212	`object not of required type`	Objekt falschen Typs
213	`disk not validated`	Diskette wird noch geprüft oder ist defekt

214	`disk write protected`	Diskette ist schreibgeschützt
215	`rename across device attempted`	Bei Rename verschiedene Laufwerke angegeben
216	`directory not empty`	Directory nicht leer (muß es vor dem Löschen sein)
218	`device not mounted`	Device (Laufwerk) nicht angemeldet
219	`seek error`	Fehler beim Suchen eines Records
220	`comment too big`	File-Kommentar zu groß (> 80 Zeichen)
221	`disk full`	Diskette voll
222	`file is protected from deletion`	File ist gegen Löschen geschützt
223	`file is protected from writing`	File ist schreibgeschützt
224	`file is protected from reading`	File ist lesegeschützt
225	`not a DOS disk`	Keine DOS-Diskette
226	`no disk in drive`	Keine Diskette im Laufwerk
232	`no more entries in directory`	Keinen Directory-Eintrag (File-Namen) mehr gefunden

Anhang

B

GenAmiga-Fehlermeldungen



Anhang: *GenAmiga*-Fehlermeldungen

GenAmiga kann eine große Zahl von Fehlermeldungen ausgeben. Die meisten davon erklären sich von selbst. Dieser Anhang bringt sie in der numerischen Reihenfolge in englisch und deutsch, ggf. mit Erklärungen.

Bitte beachten Sie, daß *GenAmiga* laufend erweitert und noch verbessert wird. Deshalb kann diese Liste nicht genau mit Ihrer Version übereinstimmen, weil neue Meldungen hinzugekommen sind.

Die Meldungen 0 bis 59 sind normale Fehler, die Nummern 60 bis 79 sind Warnmeldungen, die Progammierfehler sein können (können mit OPT W- unterdrückt werden), und alles ab 80 aufwärts sind fatale Fehler, die zum Abbruch des Assemblierens führen.

Nr.	Meldung	Bemerkung
1	`unrecognized instruction`	Unbekannter Befehl oderMakro-Name
2	`undefined label`	Label nicht definiert
3	`extra label`	Es erschien ein Label in Pass2, nicht aber in Pass 1
4	`label defined twice`	Label zweimal definiert
5	`phasing error`	Phasenfehler zwischen Pass 1 u.2, Wert eines Labels zwischen Pass 1 und 2 geändert oder Label doppelt
6	`garbage following instruction`	Nach Befehlswort folgt Unsinn, Fehler in Operand(en)
7	`space expected`	Erwarte Leerstelle zwischen Feldern
8	`immediate data expected`	Erwarte Konstante #n
9	`relative not allowed`	Relative Adressierung verboten

10	`comma expected`	Erwarte Komma
11	`data expected`	Erwarte Daten
12	`data too large`	Datentyp zu groß, Operand hat mehr Bits als Typ erlaubt
13	`hex digit expected`	Erwarte Hex-Zeichen
14	`addressing mode expected`	Erwarte Adressierungsart
15	`binary digit expected`	Erwarte 0 oder 1
16	`) expected`	Erwarte)
17	`register expected`	Erwarte Register
18	`unrecognized addressing mode`	Adressierungsart nicht bekannt
19	`. expected`	Erwarte '.'
20	`W or L expected`	Erwarte W oder L nach einem Index-Register
21	`addressing mode not allowed`	Adressierungsart nicht erlaubt
22	`Byte size not allowed`	Type Byte nicht erlaubt
23	`address register expected`	Erwarte Adreßregister, Ihre Adressierungsart ist ungültig
24	`string too long`	String zu lang, max. 126 erlaubt
25	`quote expected`	Erwarte " oder '
26	`.S or .L only`	Nur .S oder .L erlaubt, bei Verzweigungen
27	`label expected`	Erwarte Label, bei Direktiven
28	`not implemented`	Nicht implementiert, wenn externe Label in Ausdruck
29	`unbalanced ENDM`	Zahl ENDMs ungleich, Zahl MACROs
30	`Macro nested too deep`	Makroschachtelung zu tief, max. 8 Ebenen
31	`unbalanced ENDC`	Zahl IFs ungleich ENDCs
32	`IF nested too deep`	IF-Schachtelung zu tief

33	`data register expected`	Erwarte Datenregister, d.h. falsche Adr.-Art
34	`.W only`	Nur .W erlaubt
35	`invalid expression`	Ungültiger Ausdruck
36	`malformed register list`	Fehler in Register-Liste, bei MOVEM
37	`long not allowed`	Typ Lang nicht erlaubt
38	`Include nested too deep`	Includes zu tief geschachtelt
39	`binary file I/O error`	Binär-File I/O-Fehler, wie z.B Diskette voll
40	`absolute not allowed`	Absolute Adressierung nicht erlaubt
41	`bad register number`	Falsche Register-Nummer
42	`illegal use of MACRO or EQUR label`	Illegale Label, MACRO oder EQUR
43	`size not allowed`	Größe (d.h. Typ) nicht erlaubt
44	`size unrecognized`	Größe (d.h. Typ) nicht erkannt
45	`unbalanced MEXIT`	MEXIT außerhalb Makro geschrieben
46	`unrecognized option`	Unbekannte Option, in OPT-Direktive
47	`divide by zero`	Division durch Null
48	`bad type combination`	Unerlaubte Typkombination, in Ausdrücken
49	`forward reference`	Vorwärts-Referenz
50	`missing ENDC`	ENDC fehlt, wird am Textende bemerkt
51	`numeric parameter only`	Nur Zahl als Parameter erlaubt, keine Labels in CNOP
52	`user error`	Vom Anwender erzeugter Fehler, mit FAIL-Direktive

53	`directive must be at start`	Direktive muß in erster Zeile stehen, z.B. OPT L
54	`not allowed unless linking`	Nur in linkbarem Programm erlaubt
55	`directive only allowed once`	Direktive nur einmal erlaubt
56	`bad external`	Falsche XDEF
57	`external not allowed`	Externes Label nicht erlaubt
58	`cannot relocate this`	Kann dazu keine Relo-Bytes erzeugen
59	`not allowed when linking`	In linkbarem Code nicht erlaubt

Warnungen

60	`short branch of zero`	Sprungdistanz ist null, wird zu NOP
61	`signed extended operand`	Vorzeichenbehafteter Operand, bei MOVEQ 128 – 255
62	`jump to odd address`	Sprung auf ungerade Adresse
63	`relative offset used`	Offset(-Label) ist relativ
64	`68020 incompatible`	68020 inkompatibel

Fatale Fehler

80	`expression MUST evaluate`	Ausdruck muß lösbar sein
81	`symbol table full`	Symboltabelle voll
82	`file I/O failed`	Fehler im File-I/O, zeigt DOS-Fehlernr.
83	`Macro label used before defined`	Makro vor Definition eingesetzt
84	`Macro table full`	Makro-Tabelle voll. Prüfen Sie, ob ein ENDM fehlt. Wenn nicht, vergrößern Sie die Tabelle durch Include oder vergrößern Sie den Textpuffer mit *GenInst*.

Anhang

C

Aufruf von System-Routinen



Anhang: Aufruf von System-Routinen

C1 Einführung

Das Amiga-Betriebssystem ist mit Sicherheit eines der hochentwickeltsten aller Computer in Massenproduktion, aber auch das komplizierteste. Das ganze System basiert auf dem Librarykonzept. Librarys sind im Prinzip Gruppen von Unterprogrammen (Funktionen für C-Programmierer) und werden über eine Indextabelle aufgerufen. Dieser Anhang will Ihnen grundsätzlich erklären, wie man Library-Routinen von Assembler aus aufruft und soll Ihnen eine Vorstellung davon geben, wofür welche Routinen eingesetzt werden können. Ein kleiner Anhang kann nicht das ganze Betriebssystem erklären, er ist nur als Einführung gedacht. Für weitergehende Informationen sind die Bücher in den Literaturhinweisen zu empfehlen.
Beachten Sie bitte, daß dieser Anhang auf der Basis von Kickstart 1.2 geschrieben wurde. Wenn es Unterschiede zu neueren Versionen geben sollte, wird darauf im Readme-File verwiesen.

C2 Libraries

Die wichtigste Library ist die Exec-Library, die unter anderem auch dafür gebraucht wird, andere Libraries zu öffnen. Wie für alle Libraries benötigt man einen Library-Basiszeiger, der in das Register A6 geladen werden muß, bevor man eine Routine der Lib aufrufen kann. Nur die Exec-Lib muß nicht geöffnet werden, um einen Basiszeiger zu erhalten; dieser Zeiger steht im Langwort ab Adresse 4. Das ist die einzige Adresse, für die garantiert ist, daß sie sich nicht ändert. Der daraus gelesene Basiszeiger kann benutzt werden, um andere Libs zu öffnen und damit deren Basiszeiger zu erhalten und so weiter. Beachten Sie, daß die meisten Libs den Basiszeiger in A6 erwarten, wenn sie korrekt funktionieren sollen. Das liegt daran, daß viele Routinen selbst wieder Routinen derselben Lib aufrufen und dabei unterstellen, daß der Basiszeiger schon in A6 ist. Parameter werden an Library-Routinen in Registern übergeben, und als allgemeine Regel gilt, daß d0/d1 und a0/a1 nach jedem Aufruf verändert sein können. Die Hauptausnahme von dieser Regel macht die Graphics-Library, worauf später noch eingegangen wird.

Mit dem *Devpac* wird eine große Anzahl von Include-Files geliefert, um einen einfachen Zugriff auf die verschiedenen Teile des Betriebssystems zu ermöglichen. Diese Include-Files enthalten Makro-Definitionen, Symbole für die Library-Offsets, Definitionen von Datenstrukturen und Symbole für Bit-Felder. Es folgt nun eine Tabelle, die die Namen der verschiedenen Komponenten zeigt und wo diese im Include-Directory zu finden sind. Darin bedeuten:

File: File mit den Makro-Definitionen und den _LVO-Offsets zum Aufruf der Library-Routinen.

Name macro: Dieses Makro besteht normalerweise aus einer DC.B-Anweisung für den Namen (in ASCII), gefolgt von einem Null-Byte.

Base pointer: Dies ist die symbolische Adresse (Label), auf der der Basiszeiger abgelegt werden sollte. Das Symbol beginnt immer mit einem Unterstrich (_), auch wenn die meisten C-Compiler dieses Zeichen nicht listen.

Calling macro: Dies ist der Name des Makros, mit dem eine bestimmte Library-Routine aufgerufen wird. Beachten Sie, daß dieses Makro A6 verändert, weil es es mit dem Basiszeiger lädt.

Library	File	Name macro	Base pointer	Calling macro
clist	libraries/clist_lib.i	CLISTNAME	_ClistBase	CALLCLIST
diskfont	libraries/diskfont_lib.i	DISKFONTNAME	_DiskfontBase	CALLDISKFONT
dos	libraries/dos_lib.i	DOSNAME*	_DOSBase	CALLDOS
exec	exec/exec_lib.i	EXECNAME	_SysBase	CALLEXEC
graphics	graphics/graphics_lib.i	GRAFNAME	_GfxBase	CALLGRAF
icon	workbench/icon_lib.i	ICONNAME	_IconBase	CALLICON
intuition	intuition/intuition_lib.i	INTNAME	_IntuitionBase	CALLINT
mathffp	math/mathffp_lib.i	FFPNAME	_MathBase	CALLFFP
mathdouble	math/mathieeedoubbas_lib.i	IEEEDOUBNAME	_MathIeeeDoubBasBase	CALLIEEEDOUB
mathtrans	math/mathtrans_lib.i	MATHTRANSNAME	_MathTransBase	CALLMATHTRANS

*Die Name Macro Definition befindet sich in file libraries/dos.i

Neu ist: expansion in libraries/expansion_lib.i. Name: EXPANSIONNAME. Basiszeiger: _ExpansionBase. Makro: CALLEXP.

Zum Beispiel wäre der passende Aufruf für die Funktion »OpenLibrary«:

```
CALLEXEC  OpenLibrary
```

Dieses Makro wird entwickelt als

```
move.l    (_SysBase).w,a6
jsr       _LVOOpenLibrary(a6)
```

Clist-Library

Das ist die »character list«-Library. Aufgrund fehlender Dokumentation kann diese Library leider nicht näher beschrieben werden.
Files: libraries/clist.i und clist_lib.i

Diskfont-Library

Dies ist die Library zum Umgang mit Fonts, die normalerweise auf der Disk sind.
Files: libraries/diskfont.i und diskfont_lib.i

DOS-Library

Diese ist eine der am einfachsten zu benutzenden Libraries und behandelt das File-I/O (Input/Output) zu Geräten einschließlich Disketten und der Console. Diese Lib hat einige kleine Besonderheiten. Bemerkenswert ist, daß Adressen in Datenregistern übergeben werden müssen und viele Zeiger vom BCPL-Type sein müssen (d.h. Adresse dividiert durch 4 auf eine Langwortgrenze justiert).
Files: libraries/dos.i, dos_lib.i und dosextens.i

Exec-Library

Dies ist die Library auf der untersten Ebene. Sie ist zum Beispiel zuständig für Speicherverwaltung, Library-Öffnen und Nachrichtenübergabe. Die Library darf niemals geöffnet werden, ihr Basiszeiger steht ab Adresse 4.
Files: exec/ables.i, alerts.i, devices.i, errors.i, exec.i, execbase.i, execname.i, exec_lib.i, funcdef.i, intializers.i, interrupts.i, io.i, libraries.i, lists.i, memory.i, nodes.i, ports.i, resident.i, strings.i, tasks.i und types.i

Graphics-Library

Diese Lib ist für alles zuständig, was auf Ihrem Monitor erscheint, zum Beispiel Zeichnen von Linien, Ausgabe von Text, Kontrolle von RastPorts, Sprites und Fonts. Beachten Sie bitte, daß in der Kickstart-Version 1.1 die Text-Routine Register D7 verändert. Dies kann auch Auswirkungen auf andere Routinen haben, besonders auf die Routine »DoIO« bei der Consol-Ausgabe.
Files: graphics/clip.i, copper.i, display.i, gels.i, gfx.i, gfxbase.i, graphics_lib.i, layers.i, rastport.i, regions.i, sprite.i, text.i, view.i

Icon-Library

Diese Lib ist zuständig für den Umgang mit Icons, die auf der Workbench angezeigt werden.
Files: workbench/icon.i und icon_lib.i

Intuition-Library

Diese Library ist die größte und zuständig für die Bedienungsoberfläche Intuition. Sie verfügt über eine sehr große Anzahl von Funktionen wie die Kontrolle von Windows, Screens, Gadgets, Requestern und dem Event-Handling. Das Hauptfile ist groß und lädt mit Include noch andere Files nach. Seien Sie nicht überrascht, wenn es eine Weile dauert, bis das alles geladen ist. Es kann sich durchaus lohnen, eine eigene Version zu erstellen, der die weniger oft benutzten Konstanten und Include-Files fehlen.
Hinweis des Übersetzers: Schauen Sie sich die Include-Files an. Es ist oft sehr wenig, was da »included« wird und lohnt kaum den Disk-Zugriff. Andererseits werden oft große Files nachgeladen, obwohl man daraus nur eine oder zwei Konstanten braucht.
Files: intuition/intuition.i und intuition_lib.i

Maths-Libraries

Es gibt drei Mathematik-Libraries, die alle auf den offiziellen Motorola-Routinen basieren. Die FFP- (Fast Floating Point = schnelles Fließkomma) Library rechnet mit einem 8-Bit-Exponenten und einer 24-Bit-Mantisse. Das Format wurde von Motorola für die 68000er entwickelt und ist kein Standard-Format. Die »IEEE double«-Library bietet doppelte Genauigkeit im IEEE-Format und die »Transenten«-Lib trigonometrische und andere Funktionen im FFP-Format.
Files: math/mathffp_lib.i, mathieeedoubbas_lib.i und mathtrans_lib.i

Generell sollten Sie *GenAmiga* im case-sensitiven Modus (wie voreingestellt) benutzen, wenn Sie die mitgelieferten Include-Files einsetzen. Beachten Sie, daß jedes Include-File automatisch alle Include-Files lädt, die es benötigt, so daß Sie sich darum nicht kümmern müssen.

C3 Beispiel-Programme

Um Ihnen den Anfang in der Programmierung von Amiga-DOS in Assembler zu erleichtern, haben wir einige Beispiel-Programme im Directory »example« vorgesehen:

demo.s

Dies ist das Programm, das für die Einführung zu Beginn diese Handbuches benutzt wurde. Es nutzt die DOS-Library und gibt einen Text im aktuellen CLI-Fenster aus.

freemem.s

Dieses Programm unter Intuition öffnet ein Fenster, in dem der freie Hauptspeicher ständig angezeigt wird, bis das Schließ-Gadget angeklickt wird. Um das Programm als zum CLI parallele Task laufen zu lassen, geben Sie das CLI-Kommand »run freemem« ein.

helloworld.s

Dies ist die Assembler-Umsetzung des C-Programms (letzte Version) »Simple Program« im Intuition-Manual, Seite 2–9. Die Umsetzung wurde in keiner Weise optimiert. So ist es zum Beispiel wesentlich effektiver, die Zuweisungen für »NewScreen« durch DC-Befehle zu ersetzen. Das Programm öffnet einen Screen, darauf ein Window und druckt dann einen Text.

C4 Konvertierung von Amiga-Assembler-Programmen

Die meisten Programme, die für den Amiga-DOS-Assembler (Metacomco) geschrieben wurden, sollten auch ohne Probleme oder mit kleineren Änderungen von *GenAmiga* assembliert werden können. Die Unterschiede sind:

1. *GenAmiga* erlaubt zur Zeit nur eine SECTION-Direktive pro File.
2. Von Labels sind die ersten 16 Zeichen signifikant.
3. Lokale Labels werden nicht unterstützt.
4. Die OFFSET-Direktive gibt es nicht, nehmen Sie statt dessen RS.
5. Bei XREF auf Konstanten (im Gegensatz zu relativen Adressen) müssen Sie XREF.L verwenden. Viele Quellprogramme setzen XREFs für die _LVO-Labels ein. Diese können Sie durch XREF.L ersetzen oder Sie löschen einfach diese XREFs und ziehen das relevante _lib.i-File ein.
6. Die Direktiven RORG,REG und DCB werden nicht unterstützt.

Die mit dem Amiga-DOS-Assembler gelieferten Include-Files sollten ohne Änderung von *GenAmiga* assembliert werden können, aber *Devpac*-Files sind vorzuziehen und deutlich schneller. Das liegt daran, daß Direktiven an Stelle von Makros eingesetzt werden, um Labels zu definieren, und alle Kommentare entfernt wurden. Die Original-Versionen finden Sie im Anhang E des »ROM Kernel Reference Manual, Volume 2«.

C5 Vergleich CLI/Workbench

Es gibt zwei Programmumgebungen auf dem Amiga, die window- und icongesteuerte Workbench und das CLI. *Devpac*-Amiga selbst läuft unter letzterem, wie auch die meisten Beispielprogramme. Der Unterschied liegt im Startup- und Exit-Code.

CLI-Start

Wenn ein Programm vom CLI aus gestartet wird, enthält Register A0 die Adresse der Kommandozeile und D0 deren Länge. Die DOS-Handles, die die Funktionen Input und Output zurückgeben, können für die Ein/Ausgabe von/zum aktuellen CLI-Fenster benutzt werden. Solche Programme enden mit einem einfachen RTS.

Workbench-Start

Wenn ein Programm von der Workbench gestartet wurde, muß es zuerst auf eine Nachricht warten und diese Nachricht (nach einem Forbid-Aufruf) zurückgeben, bevor es mit RTS endet. Die DOS-Funktionen Input und Output geben ungültige Handles zurück, weshalb Sie ein eigenes Fenster für die Ein/Ausgabe öffnen müssen.

Lösungen

Die unterschiedlichen Start-Sequenzen sind im Kapitel 2, Teil IV des »ROM Kernel Manual Volume 1« detailliert beschrieben. Dazu gehört auch der Assembler-Quelltext für den Startup-Code von C-Programmen (Rev. 2, Seite 4-36). Beachten Sie, daß in diesem Listing ein Fehler steckt. In der Routine »openDOS« muß vor dem Aufruf von »OpenLibrary« ein »moveq #0,d0« eingefügt werden. Andernfalls kann eine Guru-Meldung erzeugt werden.
Eine Skeleton-Version dieses Programms für Assembler-Programmierer finden Sie im File »misc/easystart.i«.Dieses File wird auch vom Beispielpro-

gramm »freemem2.s« eingezogen. Das Include-File sollte ziemlich zu Beginn Ihres Programms eingefügt werden. Es erledigt die Nachrichtenübergabe so, daß das Programm von der Workbench aus gestartet werden kann. Natürlich benötigen Sie dazu ein Icon, das mit *IconEd* erstellt werden kann.

C6 Andere 680xx-Prozessoren

Wenn Sie kommerzielle Programme für den Amiga schreiben, sollten Sie bedenken, daß einige Anwender einen 68010 oder 68020 anstatt des üblichen 68000 einsetzen. Das einzige Problem dabei ist der »MOVE SR«-Befehl, der beim 68010 und 68020 privilegiert ist. Zur Lösung des Problems bietet die Exec-Library die Routine GetCC, welche die Condition-Codes im Register D0 zurückgibt. Sie gebraucht keine anderen Register und braucht (so der Ist-Stand) auch nicht den Library-Basiszeiger in A6. Um zum Beispiel die Condition-Codes auf den Stack zu bringen, schreiben Sie anstatt

```
move      sr,-(sp)
```

besser

```
movea.l   4.w,a0          hole Basiszeiger
jsr       _LVOGetCC       Aufruf von GetCC
move.w    d0,-(sp)        CCs auf den Stack
move      d0,ccr          und Codes setzen
```

Die Routine nutzt A0 als Arbeitsregister. Am besten schreiben Sie das als Makro, um ein versehentliches Ändern des Condition-Codes zu vermeiden (*movea* ändert sie nicht).

Der einzige weitere Unterschied ist der wesentlich größere Stackbereich, den die 68010/68020-Prozessoren im Falle von Exceptions füllen. Doch das dürfte nur Debugger und ähnliche Programme betreffen.

Anhang

D

Gebrauch des CLI



D Anhang: Gebrauch des CLI

D1 Einführung

Wie die meisten Programmentwicklungs-Tools, ist auch *Devpac*-Amiga für den Einsatz unter dem CLI (Command Line Interface) vorgesehen. Wenn Sie sich die Diskette von der Workbench aus ansehen, werden Sie deshalb wenige (oder keine) Icons sehen.
Devpac-Amiga wird mit seiner eigens modifizierten Workbench-Diskette geliefert, die die Workbench-Bedienungsoberfläche nicht lädt, sondern direkt ins CLI geht.

D2 Files, Laufwerke und Directories

Amiga-DOS-Files können auf Diskette, der RAM-Disk oder auf einer Hard-Disk gespeichert werden. Weil Disketten 800K Daten enthalten können und Hard-Disks sogar noch mehr, erlaubt Amiga-DOS Directories. Das sind mit Namen bezeichnete Sektionen auf den Disks, die Files oder Directories enthalten können. (Wenn Sie die Workbench benutzen, heißen Directories Drawers (Schubladen.) Um ein File innerhalb eines Directory anzusprechen, wird das »/«-Zeichen benutzt. Wenn Sie also ein File mit dem Namen »test.s« editieren wollen, der im Directory »source« steht, gilt das Kommando

```
genam source/test.s  [Return]
```

Beachten Sie, daß das CLI Groß- und Kleinbuchstaben nicht unterscheidet, weshalb die Eingabe

```
GenAm Source/Test.s  [Return]
```

dasselbe bewirkt.
Der Aufruf von *GenAmiga* ist ähnlich dem der meisten CLI-Kommandos. Sie bestehen aus einem Kommando (hier *GenAm*), dem optional noch etwas folgt, das man Kommandozeile nennt.

Wenn Sie wissen wollen, welche Files auf der Disk sind, geben Sie ein

```
dir [Return]
```

was alle Files zeigt. (dir heißt zeige DIRectory.) Amiga-DOS kann dafür länger brauchen als andere Systeme, seien Sie also geduldig. Wenn es Directories auf der Disk gibt, werden diese zuerst angezeigt, und zwar gefolgt von »(dir)«. Darauf folgen die File-Namen in alphabetischer Reihenfolge.
Unglücklicherweise kümmert sich Amiga-DOS nicht um die Schreibweise der File-Namen hinsichtlich Groß-/Kleinbuchstaben, so könnten Sie sich wundern, warum das im Ausdruck trotzdem unterschieden wird. Die Antwort ist, daß der Name in der Schreibweise benutzt wird, die beim Anlegen des Files oder Directories gewählt wurde.
Das CLI greift immer auf das aktuelle Laufwerk und Directory zu. Mit dem CD-Kommando können Sie den jeweiligen Stand erfragen und ändern. Nach dem Booten mit einer Kopie der *Devpac*-Diskette wird die aktuelle Disk in DF0: sein (DF0: = internes, DF1: = externes Laufwerk.) Sie können das testen mit

```
cd [Return]
```

Tippt man CD alleine, wird immer das aktuelle Directory angezeigt. Wenn Sie in ein anderes Directory wechseln wollen, zum Beispiel in das mit den Beispielprogrammen, müssen Sie CD nur noch den Namen folgen lassen, d.h.

```
cd examples [Return]
```

Wenn Sie jetzt »dir« eingeben, sehen Sie eine Liste von Files. Geben Sie nun nur cd ein, sollten Sie »DF0:examples« sehen.

Um sich Textfiles anzusehen, was hier alle sind, die ».s« enden, können Sie zum Beispiel eingeben

```
type demo.s [Return]
```

Das zeigt den Inhalt von »demo.s« auf dem Schirm. Um das (durchlaufende) Bild anzuhalten, können Sie die rechte Maustaste drücken; um das Listing vorzeitig abzubrechen, geben Sie Ctrl-C ein.

Um in das vorherige Directory zurückzukommen, geben Sie ein

```
cd df0: [Return]
```

oder alternativ

```
cd : [Return].
```

Der Doppelpunkt alleine bezeichnet das Directory auf der obersten Ebene (Root-Directory). Wenn wir uns nun »demo.s« noch einmal ansehen wollen, können wir wieder das Directory wechseln, aber einfacher ist es, das Directory zusammen mit dem File-Namen anzugeben, also

```
type examples/demo.s  [Return]
```

Beachten Sie die Art, wie der Schrägstrich (/) eingesetzt wird.

D3 Amiga-DOS-Wildcards

Wildcards sind Platzhalter für beliebige Zeichen oder Zeichenfolgen übersetzen. Amiga-DOS hat unglücklicherweise eine völlig unterschiedliche Lösung für Wildcards als andere Betriebssysteme. Zugegebenermaßen hat man damit viel mehr Möglichkeiten, aber leicht verständlich sind sie nicht. Hier eine Zusammenfassung:

?	ein einzelnes Zeichen
%	Nullstring
#<p>	ein- oder mehrfaches Auftreten des Musters <p>
<p1>|<p2>	Entweder Muster p1 oder Muster p2
()	faßt Muster zusammen

Im allgemeinen braucht man davon das »?«, was sich von selbst erklärt, und »#?«, was dem »*« von MS-DOS und CP/M entspricht.

D4 Geräte-Namen

Alle I/O-Geräte des Amiga haben Namen, über die Sie sie in CLI-Kommandos und in Programmen ansprechen können. Es sind kurze Namen, die mit einem Doppelpunkt enden. Tatsächlich haben Sie schon zwei davon kennengelernt, nämlich die beiden Floppy-Laufwerke DF0: und DF1:. Hier ist der Rest:

RAM:

Dies ist die RAM-Disk, die wie ein normales Laufwerk eingesetzt werden kann. Sie ist allerdings sehr viel schneller, weil die Daten direkt im Hauptspeicher aufgezeichnet werden. Ihr Nachteil ist, daß die Daten verloren sind, wenn der Rechner ausgeschaltet wird, ein Reset erfolgt oder Ihr Programm etwas verrückt spielt. Die RAM-Disk ist dynamisch, d.h., sie belegt immer nur soviel Speicher, wie für das jeweilige Datenvolumen benötigt wird.

Anmerkung zu Kickstart 1.1: Wenn man die RAM-Disk bis zum letzten Byte füllt, stürzt das System ab, weil es ohne Erfolg (mangels RAM) versucht, die Meldung »Disk voll« auszugeben.

PAR:

Dies ist der Parallel-Druckerport, der normalerweise nur für die Ausgabe (Output) eingesetzt wird.

SER:

Dies ist die serielle Schnittstelle, die für Ein- und Ausgabe benutzt werden kann, normalerweise zu Druckern und Modems.

PRT:

Dies ist der allgemeine Druckerport, der mit »Preferences« konfiguriert wird. Hierfür muß das Programm nicht wissen, an welchem Port (PAR: oder SER:) der Drucker hängt oder welcher Druckertyp angeschlossen ist.

NIL:

Dies ist ein Dummy-Gerät, auf das man Ausgaben ins Nichts schicken kann. Der Versuch von NIL: zu lesen, generiert die Fehlermeldung »end of file«.

CON:

Dies ist die Console. Damit wird ein Window für die Tastatur-Ein-/Ausgabe geöffnet. Die Syntax lautet:

```
CON:x/y/Breite/Höhe/Titel
```

RAW:

Das ist, um das Amiga-DOS-Handbuch zu zitieren, »für den fortgeschrittenen Anwender gedacht. Experimentieren Sie nicht mit RAW:«.

D5 Amiga-DOS-Befehle

Um Ihnen beim Einsatz des CLI zu helfen, folgt nun eine kurze Beschreibung der am meisten verwendeten Befehle zusammen mit Beispielen. Es ist keine vollständige Beschreibung. Wenn Sie weitere Details benötigen, empfehlen wir, auf weiterführende Literatur zurückzugreifen.

[] bezeichnen erforderliche Parameter, < > optionale. Die meisten Kommandos erlauben die Umleitung der Ausgabe mit »>«, so kann zum Beispiel ein Directory-Listing in einen File geschrieben werden mit

```
dir >allfiles.txt
```

D5.1 ASSIGN <Name> <Directory>

Dies erlaubt Ihnen, spezielle Namen – besonders für Directories – zu vergeben. Beim Start werden vom System die folgenden Zuweisungen gesetzt:

Name	Voreinstellung	Anwendung/Zweck
C:	boot:c	CLI sucht hier Kommandos
L:	boot:l	System-Handlers
S:	boot:s	Startup-sequence
LIBS:	boot:libs	Libraries nicht im ROM
DEVS:	boot:devs	Geräte-Treiber
FONTS:	boot:fonts	Zeichensätze nicht im ROM
SYS:	boot:	Boot-Diskette

Genausogut, wie Sie das ändern können, können Sie auch eigene Zuweisungen machen. Wenn Sie zum Beispiel einfacher auf das Graphics-Include-File auf der Diskette im externen Laufwerk zugreifen wollen, schreiben Sie

```
assign graf: df1:include/graphics  [Return]
```

Dann können Sie unabhängig vom aktuellen Directory auf Files in dieser Art zugreifen:

```
type graf:rastport.i
```

Um alle Zuweisungen aufzulisten, geben Sie nur »assign« ein, und um eine Zuweisung aufzuheben, wiederholen Sie sie, ohne das Directory (den Pfad) anzugeben.

D5.2 CD <Directory>

Dies ändert entweder das Directory, wenn dem CD ein Parameter folgt, oder zeigt das aktuelle Directory an (wenn nur CD eingegeben wird).

D5.3 COPY [Altname] <TO> [Neuname] <ALL> <QUIET>

COPY kopiert Files einzeln oder in Gruppen oder auch ein ganzes Directory. Mit der ALL-Option werden Directories kopiert. Die QUIET-Option unterdrückt das Listing der Filenamen. Jeder existierende File wird ohne Rückfrage überschrieben. Am besten zeigt man das anhand von Beispielen:

```
copy examples/demo.s to ram:demo.s [Return]
```

kopiert ein einzelnes File.

```
copy examples to df1:genexamples all [Return]
```

kopiert ein komplettes Directory in ein anderes (das bereits existieren muß).

```
copy include to df1:include all [Return]
```

kopiert das Include-Directory sowie alle Directories darin auf das externe Laufwerk und legt dabei die neuen Directories an, soweit nötig.

```
copy #?.info ram:
```

kopiert alle Files, die mit ».info« enden, auf die RAM-Disk.

D5.4 DATE <Datum> <Zeit>

DATE zeigt oder setzt das Datum und/oder die Uhrzeit. Das Format für das Datum ist TT-MMM-JJ, das für die Zeit HH:MM. Es werden beide zusammen oder einzeln gesetzt; folgt kein Parameter, wird der Ist-Stand angezeigt. Beispiel:

```
date 25-jan-87 12:00
```

D5.5 DELETE [Name] <Name> <Name>.. <ALL>

DELETE löscht Files einzeln oder in in Gruppen oder auch ein ganzes Directory. Wenn Sie nur ein Directory angeben, wird es nur gelöscht, wenn es leer ist. Die ALL-Option löscht aber vorab den Inhalt, z.B.

```
delete demo.s [Return]
delete examples all [Return]
```

D5.6 DIR <Name>

Listet das aktuelle Directory, wenn <Name> fehlt, sonst das benannte Directory.

D5.7 DISKCOPY [drive] TO [drive] <NAME name>

Das Kommando kopiert eine Diskette komplett auf eine andere. Die Zieldiskette muß nicht formatiert sein, alle darauf schon vorhandenen Files werden gelöscht. Wenn Sie für beide Laufwerke denselben Namen angeben, werden Sie jeweils zum Diskettenwechsel aufgefordert. Es wird empfohlen, auf der Quelldiskette den Schreibschutz zu aktivieren, bevor Sie das Programm starten. Sonst könnten versehentlich Daten verlorengehen. Per Voreinstellung bekommt die Ziel-Diskette den Namen des Originals, das kann aber mit der NAME-Option geändert werden.

```
diskcopy df0: to df1: [Return]
diskcopy df0: to df0: name "Backup" [Return]
```

D5.8 ECHO <String>

Das Kommando gibt <String> auf dem Schirm aus. Es ist hauptsächlich in der Startup-Sequence (siehe später) und in Execute-Batch-Files von Nutzen. Beispiel:

```
echo "Devpac Amiga Workbench disk" [Return]
```

D5.9 ENDCLI

Damit verlassen Sie das aktuelle CLI, aber Vorsicht!!!!!
Sind Sie nämlich schon im letzten CLI (CLI 1) und haben Sie nicht die Workbench geladen (mit loadwb), kommen Sie nicht mehr zurück. Sie können dann nur noch neu booten. Der Befehl ist zum Verlassen von CLIs gedacht, die Sie vorher mit NEWCLI eingerichtet hatten.

D5.10 EXECUTE [Name] <Argumente>

Für häufig verwendete Kommandofolgen im CLI kann man ein Text-File anlegen, in dem die Kommandos aufgeführt werden. Ruft man dieses Text-File mit EXECUTE auf, so werden alle Befehle nacheinander abgearbeitet. Parameter werden an Stelle von Platzhaltern eingesetzt. Dies funktioniert ähnlich wie bei

den Makros von *GenAmiga*. Heißt das Text-File zum Beispiel »doit« und enthält diese Anweisungen.

```
.key file/a
copy <file> to ram:
copy ram:<file> to prt:
delete ram:<file>
```

dann bewirkt

```
execute doit demo.s [Return]
```

daß das File demo.s auf die RAM-Disk kopiert wird, von dort auf den Drucker (und somit gedruckt wird) und schließlich auf der RAM-Disk gelöscht wird. Das ».key« spezifiziert einen Parameter (hier file), und »/a« heißt, daß der Parameter immer (always) erforderlich ist.

D5.11 FAULT <Zahl>

Dieses Kommando gibt die Bedeutung (als Text) einer Fehlernummer von Amiga-DOS auf dem Schirm aus.

D5.12 FORMAT DRIVE [drivename] NAME [diskname]

FORMAT initialisiert (formatiert) eine Diskette, wobei ihr bisheriger Inhalt zerstört wird. »drivename« kann DF0: oder DF1: sein, »diskname« ist der Name der Diskette. Beispiel:

```
format drive df1: name "My backups" [Return]
```

D5.13 INFO

INFO bringt folgende Informationen auf den Schirm: Status jedes Laufwerks und jeder montierten Diskette einschließlich deren Namen und des freien Bereichs in Blöcken. Ein Block hat 512 Bytes, womit die Hälfte der Blockzahl die Größe in Kbyte angibt.

D5.14 JOIN [file1] <file2 file3 u.s.w.> AS [newfile]

JOIN kopiert bis zu 15 Files zusammen in ein neues File. z.B.

```
join part1  part2 AS allparts [Return]
```

D5.15 LIST <name>

LIST wirkt wie DIR, nur daß auch noch die File-Größe und das Entstehungsdatum angezeigt werden.

D5.16 LOADWB

Dieses Kommando lädt die Workbench und bringt deren Screen hinter das CLI-Window. Die Standard-Workbench-Disketten haben diese Anweisung in der »startup-sequence«, so daß die Workbench automatisch geladen wird. Auf der *GenAmiga*-Diskette fehlt das, womit Speicher gespart wird.

D5.17 MAKEDIR [directory]

Damit wird ein neues Directory angelegt. Es darf kein File oder Directory gleichen Namens bereits existieren.

D5.18 NEWCLI <window>

Damit wird ein neues CLI angelegt, das als parallele Task läuft. Zusätzlich kann das Window wie bei CON: (siehe D4) spezifiziert werden.

D5.19 RENAME [oldfile] <TO> [newfile]

Dieses Kommando ändert den Namen eines Files. Es kann auch benutzt werden, um ein File in ein anderes Directory zu bringen. Beispiele:

```
rename demo.s to demo.bak [Return]
rename include/misc/start.i to examples/startup.s
```

D5.20 RUN [command]

RUN erlaubt Kommandos oder Programme parallel (gleichzeitig) ablaufen zu lassen. Dazu wird ein neues CLI (ohne Fenster) geöffnet, dem das Kommando übergeben wird. Nach dessen Ende wird das CLI wieder entfernt. Wenn Sie eine Folge von Kommandos so laufen lassen wollen, müssen Sie ein Pluszeichen (+) an das Ende jeder Zeile setzen. Beispiele:

```
run copy demo.s to prt: [Return]
```

druckt das File demo.s im Hintergrund,

während

```
run copy demo.s to prt:+ [Return]
echo "File ist gedruckt" [Return]
```

das gleiche tut, dann aber noch die Meldung bringt.

D5.21 STACK <Größe>

Dieses Kommando setzt die Stack-Größe für alle Tasks, die unter dem CLI laufen, oder zeigt nur die Ist-Größe an, wenn man »Größe« wegläßt. Voreingestellt sind 4000 Bytes, was für alle CLI-Kommandos und unsere Programme ausreicht, allerdings mit einer Ausnahme. Beim DIR-Kommando reichen bei sehr tiefgeschachtelten Directories 4000 Bytes nicht aus. Außerdem gibt es andere Programme (wie einige C-Compiler), die einen größeren Stack benötigen. Das sollte aber im Handbuch dieser Programme dokumentiert sein.

D5.22 TYPE [name] <OPT N> <OPT H>

Das Kommando wird normalerweise benutzt, um ein Textfile auf dem Schirm auszugegeben, wobei das Display mit der rechten Maustaste angehalten werden kann. Wenn Sie ein Nicht-Text-File ausgeben, kann es vorkommen, daß der Zeichensatz umgeschaltet wird und dann nur noch Unsinn erscheint. Um das zu korrigieren, drücken Sie

```
[Return] Ctrl-O [Return]
```

Die N-Option fügt Zeilennummern hinzu, mit der H-Option erreichen Sie den üblichen Hex-Dump mit gleichzeitiger ASCII-Darstellung (nicht druckbare Zeichen als Punkte).

D5.23 WHY

WHY (warum) kann nach einer CLI-Fehlermeldung eingegeben werden und bringt dann noch weitere Erklärungen.

D6 Startup-Sequence

Wenn Sie von einer Workbench-Diskette booten, wird der File »s/startup-sequence« geladen, dann werden alle darin aufgeführten Kommandos ausgeführt. Eines dieser Kommandos heißt »loadwb« (lade Workbench). Genau

diese Anweisung fehlt auf der *GenAmiga*-Diskette, außerdem auch der sonst vorhandene ENDCLI-Befehl. Damit bleiben Sie im CLI. Wenn Sie das File editieren wollen, können Sie das mit *GenAmiga* tun. Geben Sie ein

```
genam s:startup-sequence [Return]
```

(s: bewirkt, daß das File unabhängig vom aktuellen Directory gefunden wird.) Sie können nun das File ändern. Die neue Sequenz wird beim nächsten Booten ausgeführt.

D7 Erstellen einer neuen CLI-Diskette

Zur Zeit wird das *Devpac* auf einer Diskette mit der Workbench-Version 1.2 geliefert, allerdings aus Platzgründen um einige Standard-Files gekürzt. Wenn Sie eine andere oder eine vollständige Version einsetzen wollen, müssen Sie eine neue CLI-Diskette anlegen.

Zuerst müssen Sie in das CLI gelangen, zum Beispiel durch Booten mit der Kopie der *GenAmiga*-Diskette. Dann erstellen Sie mittels DISKCOPY eine Kopie der neuen Workbench-Diskette. Auf dieser müssen Sie die »startup-sequence« so ändern, daß Sie nach dem Booten direkt in das CLI gelangen. Anwender mit zwei Disketten-Laufwerken sollten dazu die neue Workbench-Disk in das externe Laufwerk tun und dann eingeben

```
genam df1:s/startup-sequence [Return]
```

Hingegen sollten Sie mit nur einem Laufwerk so vorgehen:

```
genam [Return]
```

Wenn *GenAmiga* geladen wurde, legen Sie die neue Workbench-Diskette ein, wählen »Load« aus dem Project-Menü und geben als File-Namen ein

```
df0:s/startup-sequence
```

Unabhängig von der Anzahl der Laufwerke sollten Sie an dieser Stelle in der Lage sein, das File zu editieren. Es enthält einige Echo-Anweisungen, die Sie vielleicht bei dieser Gelegenheit gleich ändern oder ergänzen, und einige sonstige Befehle. Wichtig sind jedoch die beiden letzten Zeilen, die so aussehen sollten:

```
LoadWB
endcli >nil:
```

Das »LoadWB« lädt die Workbench, was Sie vielleicht so lassen, aber die Zeile »endcli >nil:« muß gelöscht werden. Sie sollten nun die Änderung mit »Save« sichern und *GenAmiga* mit »Quit« verlassen. Nachdem alle Disk-Aktivitäten beendet sind, lösen Sie mit der Tastenkombination Ctrl und beiden Amiga-Tasten einen Reset aus und booten dann mit der neuen Workbench-CLI-Disk. Nach dem Booten sollten Sie im CLI sein. Nun müssen Sie die relevanten Files von der *GenAmiga*-Diskette auf die neue Workbench-Diskette kopieren.

Kopieren von *GenAmiga*/System mit zwei Laufwerken

Legen Sie die *GenAmiga*-Diskette in das externe Laufwerk und geben dann ein

```
copy df1:c/genam c: [Return]
copy df1:c/monam c: [Return]
copy df1:c/geninst c: [Return]
```

Mit zwei Laufwerken ist es das beste, die Workbench-Disk mit den Kommando-Files im internen Laufwerk und die Quelltexte mit den Include-Files im externen Laufwerk zu halten.

Kopieren von *GenAmiga*/System mit einem Laufwerk

Wir müssen drei Programme und zwei komplette Directories von einer Diskette auf eine andere kopieren und empfehlen daher den Weg über die RAM-Disk.

Zuerst sollten Sie mit Ihrer neuen Workbench-CLI-Disk booten. Weil auf dieser Diskette wenig Platz ist, müssen Sie erst welchen schaffen. Wenn Sie nicht vorhaben, die Workbench zu nutzen, wird folgende Prozedur empfohlen:

```
delete libs/translator.library
delete devs/narrator.device
delete trashcan all
delete empty all
delete fonts all                  evtl. Fehlermeldung macht nichts
delete utilities all              kann auch unwichtigen Fehler
                                  melden
delete #?.info
delete clock
delete ram:#?                     RAM-Disk leeren
```

`copy c:copy to ram:`	
`copy c:assign to ram:`	
`copy c:dir to ram:`	
`copy c:cd to ram:`	
`copy c:makedir to ram:`	alle nötigen Kommandos auf RAM-Disk
`assign c: ram:`	nun werden sie da auch geholt

(Vergessen Sie nicht, Return nach jeder der oben genannten Zeilen einzugeben, dito bei den nun folgenden.) An dieser Stelle legen Sie die *Devpac*-Amiga-Diskette ein und geben dann

`cd df0:`	
`copy c/genam to ram:`	
`copy c/monam to ram:`	
`copy c/genist to ram:`	
`makedir ram:examples`	für Beispiel-Programme
`copy examples to ram:examples all`	

ein.

Nun legen Sie die Workbench-Disk ein und tippen:

```
cd df0:
copy ram:genam to c
copy ram:monam to c
copy ram:geninst to c:
makedir examples
copy ram:examples to examples all
delete ram:examples all
delete ram:genam
delete ram:monam
delete ram:geninst
```

Legen Sie die *Devpac*-Amiga-Disk ein und geben:

```
cd df0:
makedir ram:include
copy include to ram:include all
```

ein.

Legen Sie wieder die Workbench-Disk ein und geben:

```
cd df0:
makedir include
copy ram:include to include all
assign c: df0:                  Kommandos wieder von Floppy
delete ram:#?                   RAM-Disk leeren
```

ein.

Das war's! Es ist eine etwas langwierige Prozedur, aber das kommt ja nur einmal vor. Wenn Sie die Beispiel-Programme (in »examples«) nicht auf die Workbench-Diskette kopieren wollen, lassen Sie die entsprechenden Befehle weg. Das spart auch einiges an wertvollem Platz.

Stichwortverzeichnis

E

F

G

H

I

J

R

S

T

U

W

X

Z

HiSoft

Devpac Assembler

deutsch

für die Amiga 500, 1000 und 2000

Devpac für Amiga 500, Amiga 1000, Amiga 2000

Ein komplettes Assembler-Entwicklungspaket für alle anspruchsvollen Amiga-Programmierer!

Mit diesem Entwicklungspaket erhalten Sie eine Reihe von Programmen, die Sie bei der Erstellung von 68000er-Assembler-Programmen und deren Umgang tatkräftig unterstützen.

Folgende Bestandteile sind enthalten:

- GenAmiga, ein Makro-Assembler mit integriertem Bildschirmeditor.

  Durch die Kombination von Assembler und Editor wird eine problemlose Eingabe und ein schnelles Assemblieren (1000 Zeilen in 6 Sekunden bei Verwendung einer RAM-Disk) gewährleistet. Der Editor unterstützt alle gängigen Standardfunktionen und arbeitet unter Intuition – es muß also nicht auf Fenstertechnik, Mausbedienung und Menüs verzichtet werden.

Der Zwei-Paß-Assembler erfüllt den Motorola-Standard. Mit hoher Geschwindigkeit können sowohl linkbare als auch sofort ausführbare Binärdateien generiert werden.

Bei Auftritt eines Fehlers erfolgt automatisch ein Rücksprung in den Editor, wo die falsche Eingabe bequem berichtigt werden kann. Zeitraubendes Nachladen und Speichern entfällt.

- MonAmiga, ein symbolischer Debugger und Disassembler. Mit MonAmiga können Sie Ihre Programme im Speicher inklusive aller Labels überprüfen und müssen sich nicht mit sechsstelligen Hex-Zahlen auseinandersetzen. MonAmiga bietet auch die Möglichkeit, einzelne Befehle schrittweise nacheinander auszuführen. Durch Programmfehler entstehende Exceptions werden vom Debugger so abgefangen, daß es selten zu den berüchtigten »Guru«-Meldungen kommt.

Devpac – Ihre professionelle Komplettlösung!

Lieferumfang:

- $3^1/_2$"-Diskette
- deutschsprachiges Handbuch

Hardware-Anforderungen:

- Amiga 500, 1000 oder 2000 mit mindestens 512 Kbyte Arbeitsspeicher
- Monitor
- mindestens ein Disketten-laufwerk.

Software-Anforderung:

Kickstart Version 1.1 oder höher.
Zu bestellen unter 51656.





DM 148,-
sFr 134,-
öS 1690,-
Unverbindliche Preisempfehlung

Hans-Pinsel-Straße 2
D-8013 Haar bei München